Approved for public release; distribution is unlimited.

Paralellization of the Naval Space Surveillance Center (NAVSPASUR) Satellite Motion Model

by

Warren E. Phipps Jr.
Captain, United States Army
B.S., United States Military Academy, West Point, NY

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

Unclassified
URITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

REPORT SECURITY CLASSIFICATION
CLASSIFIED

1b. RESTRICTIVE MARKINGS

SECURITY CLASSIFICATION AUTHORITY

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution is unlimited

DECLASSIFICATION/DOWNGRADING SCHEDULE

ERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

NAME OF PERFORMING ORGANIZATION
val Postgraduate School

6b. OFFICE SYMBOL
MA

7a. NAME OF MONITORING ORGANIZATION

ADDRESS (City, State, and ZIP Code)
nterey, CA 93943-5000

7b. ADDRESS (City, State, and ZIP Code)

NAME OF FUNDING/SPONSORING ORGANIZATION

8b. OFFICE SYMBOL

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

TITLE (Including Security Classification)
allelization of the Naval Space Surveillance Center (NAVSPASUR) Satelite Motion Model

ERSONAL AUTHOR(S)
PPS, Warren Edward Jr.

TYPE OF REPORT	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day)	15. Page Count
ster's thesis		1992,June	127

SUPPLEMENTAL NOTATION
e views expressed in this thesis are those of the author and do not reflect the official policy or position of the partment of Defense or the U.S. Government.

COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
IELD	GROUP	SUB-GROUP	Parallel processing, Satellite Motion Model, Hypercube

ABSTRACT (Continue on reverse if necessary and identify by block number)

The Naval Space Surveillance Center (NAVSPASUR) uses an analytic satellite motion model based on the Brouwer-Lyddane theory to assist in tracking over 6000 objects in orbit around the Earth. The satellite motion model is implemented by a Fortran subroutine, PPT2. Due to the increasing number of objects required to be tracked, NAVSPASUR desires a method to reduce the computation time of this satellite motion model. Parallel computing offers one method to achieve this objective. This thesis investigates the parallel computing potential of the NAVSPASUR model using the Intel iPSC/2 hypercube multi-computer. The thesis develops several parallel algorithms for the NAVSPASUR satellite motion model using the various methods of parallelization, applies these algorithms to the hypercube, and reports on each algorithm's potential reduction in computation time. A diskette containing the Fortran software developed is available upon request from neta@boris.math.nps.navy.mil.

DISTRIBUTION/AVAILABILTIY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC

1a. REPORT SECURITY CLASSIFICATION
Unclassified

NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
f. Beny Neta	(408)646-2235	MA/Nd

DD Form 1473, JUN 86

Previous editions are obselete.
S/N 0102-LF-014-6603

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ABSTRACT

The Naval Space Surveillance Center (NAVSPASUR) uses an analytic satellite motion model based on the Brouwer-Lyddane theory to assist in tracking over 6000 objects in orbit around the Earth. The satellite motion model is implemented by a Fortran subroutine, PPT2. Due to the increasing number of objects required to be tracked, NAVSPASUR desires a method to reduce the computation time of this satellite motion model. Parallel computing offers one method to achieve this objective. This thesis investigates the parallel computing potential of the NAVSPASUR model using the Intel iPSC/2 hypercube multi-computer. The thesis develops several parallel algorithms for the NAVSPASUR satellite motion model using the various methods of parallelization, applies these algorithms to the hypercube, and reports on each algorithm's potential reduction in computation time. A diskette containing the Fortran software developed is available upon request from neta@boris.math.nps.navy.mil.

TABLE OF CONTENTS

I. INTRODUCTION 1

II. NAVSPASUR SATELLITE MOTION MODEL 3

A. OVERVIEW 3

B. THEORY 5

1. Brouwer's Model 6

2. Lydanne's Modifications 17

3. NAVSPASUR Modifications 20

a. Atmospheric Drag 20

b. Near Critical Inclination 23

C. PPT2 24

III. PARALLEL COMPUTING 34

A. OVERVIEW 34

1. Definition 34

2. Classification of Parallel Computers . . . 35

a. Type Classifications 35

b. Architectural Classifications 37

c. Topological Classifications 38

3. Measurements of Performance 38

B. INTEL iPSC/2 HYPERCUBE 43

C. METHODS OF PARALLELIZATION 44

1. Vectorization 44
2. Distributing Computations 45
a. Control Decomposition 46
b. Domain Decomposition 47
3. Improving Performance 48
a. Load Balance 48
b. Communication to computation ratio . . 48
c. Sequential Bottlenecks 49

IV. PARALLELIZATION OF PPT2 50
A. VECTORIZATION 50
B. DISTRIBUTION OF COMPUTATIONS 52
1. Control Decomposition -- P^3T-4 54
a. Algorithm 54
b. Assessment 59
(1) Results 59
(2) Improvements 63
2. Domain Decomposition -- P^3T 66
a. Algorithm 67
b. Assessment 69
(1) Results 69
(2) Improvements. 73

V. CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH . 80

APPENDIX A: INPUT/OUTPUT OF PPT2 83

APPENDIX B: INTEL iPSC/2 SPECIFICATIONS 88

APPENDIX C: P^3T-4 SOURCE CODE LISTING 90

APPENDIX D: P^3T SOURCE CODE LISTING 110

LIST OF REFERENCES 115

INITIAL DISTRIBUTION LIST 117

LIST OF FIGURES

Figure 2.1 Classical Orbital Elements 8

Figure 2.2 Perturbative Accelerations on Satellites . . 21

Figure 2.3 Near Critical Inclination 25

Figure 3.1 Two-dimensional meshes 39

Figure 3.2 Hypercubes of dimension zero through four . 40

Figure 4.1 Hierarchy of NAVSPASUR Formulas 56

Figure 4.2 Hierarchy of NAVSPASUR Formulas (continued) 57

Figure 4.3 **P^{3}T-4** Algorithm 60

Figure 4.4 **PPT2** Execution Times 61

Figure 4.5 **P^{3}T-4** Execution Times 62

Figure 4.6 **P^{3}T-2** Algorithm 65

Figure 4.7 **P^{3}T** Algorithm 70

Figure 4.8 **P^{3}T** Execution Times 71

Figure 4.9 Estimated Execution Time of **P^{3}T** for Various Hypercube Sizes 78

Figure 4.10 Estimated Speedup and Efficiency of **P^{3}T** for Various Hypercube Sizes 79

ACKNOWLEDGMENT

To my advisor, Professor Beny Neta, I wish to express my deep appreciation for his continual guidance and unwavering confidence in my abilities. To Professor Danielson, I wish to thank for inspiring my study of orbital mechanics and the mathematics of satellite position prediction.

Finally, I wish to thank my wife and son, Sylvia and Brian. Without their unconditional love, untiring patience, and complete understanding, this thesis would not be possible.

I. INTRODUCTION

The Naval Space Surveillance Center (NAVSPASUR) currently tracks daily over 6000 objects in elliptical orbits around the Earth. To assist in identification and tracking of these objects in orbit, NAVSPASUR uses an analytic satellite motion model implemented in the Fortran subroutine, **PPT2**. This subroutine predicts an artificial satellite's position and velocity vectors at a selected time to aid in the tracking endeavor. Several calls to the subroutine may be required to aid in the identification of one object.

With the current increase in space operations, the number of objects necessary to be tracked is expected to increase substantially. Subroutine **PPT2** provides orbit prediction within an adequate response time for the current number of tracked objects in space. However, a substantial increase in the number of objects will cause the use of **PPT2** on a serial computer to become less responsive and computationally burdensome. Additionally, if there exists a desire to increase the accuracy of the NAVSPASUR model, the resulting subroutine would require even more computing resources and make achieving results even more time consuming.

Parallel computing offers one option to decrease the computation time and achieve more real-time results. Use of parallel computers has already proven to be beneficial in

reducing computation time in many other applied areas. Parallel computing offers an opportunity to both increase the efficiency of the current model or reduce the computational burden of a more accurate future model.

The ultimate objective of this thesis is to quantitatively determine the parallel computing potential of the current NAVSPASUR analytic model and determine the subsequent reduction in computer time if the model is applied to a hypercube multicomputor. The following chapter provides a description of the NAVSPASUR satellite model and outlines the algorithm used by the Fortran subroutine, **PPT2**. Chapter III provides an overview of parallel processing and discusses the methods to decompose a serial algorithm to be applied to the hypercube. In Chapter IV, the two methods of decomposing the analytic model are presented with their respective success in reducing computation time. The last chapter of this thesis provides conclusions and suggestions for future research.

II. NAVSPASUR SATELLITE MOTION MODEL

A. OVERVIEW

The satellite motion model, adopted by NAVSPASUR and implemented in subroutine **PPT2**, is a general perturbations variation of elements model of artificial satellite motion around the Earth. Given a set of a satellite's "mean" orbital elements at a given epoch, the model predicts the state (position and velocity) vector at a future time. The model considers perturbing accelerations caused by atmospheric drag, oblateness of the Earth, and asymmetry of the Earth's mass about the equatorial plane. This model ignores perturbations due to longitudinal variation in the Earth's gravitational potential and the influence of other celestial bodies such as the Moon or the Sun.[1]

Satellite motion models can be classified by the technique used to integrate a satellite's equations of motion and the method to describe the variation of the satellite's orbit in reaction to the perturbing forces. The two primary techniques to solve satellite's equations of motion are *general perturbations* and *special perturbations*. *General Perturbation*

---

[1]Although the NAVSPASUR model, implemented by PPT2, neglects the longitudinal variation to the Earth's gravitational potential, a correction for this variation can be made within PPT2 by a call to a second subroutine, LUNAR.

techniques involve an analytic integration of the perturbing accelerations, while *special perturbation* techniques involve the direct numerical integration of the equations of motion to include all perturbing accelerations. Typically, general perturbation techniques are more difficult and not as accurate as special perturbation techniques. However, special perturbation techniques, in general, provide this increase in accuracy at a cost of several orders of magnitude of computing resources (Bate, 1971, pp. 385 - 414). A third technique, now gaining in popularity, is a semi-analytic technique. This technique is combination of both the general and special perturbation techniques. The NAVSPASUR model, a general perturbations model, solves the satellite's equations of motion by using series solutions to the ordinary differential equations.

Within these techniques exists two methods to describe the variations to a satellite's orbit. One method, *variation of elements*, describes variations to the orbit in terms of changes in the osculating orbital elements with respect to time. The other method, *variation of coordinates*, selects a coordinate system and describes variations to position and velocity in this coordinate system with respect to time. The disadvantage of the variation of coordinates method is that the solution provides no immediate insight to the geometry of the orbit (Danby, 1989, p. 319). Using the variation of elements method, the NAVSPASUR model describes the variations

to an orbit in terms of changes to the classical orbital elements with respect to time.

B. THEORY

The NAVSPASUR model is based on a theory developed in 1959 by Dirk Brouwer of Yale University (Brouwer, 1959, pp. 378 - 397) and modified by R. L. Lyddane of the U. S. Naval Weapons Laboratory in 1963 (Lyddane, 1963, pp. 555 - 558). This theory considers an Earth's gravitational potential significantly more involved than the gravitational potential used in the classical Kepler model of idealized satellite motion. The classical model assumes a perfectly spherical Earth and the gravitational potential may be expressed by

$$U=\frac{\mu}{r} \tag{2.1}$$

where μ is the gravitational parameter and r is the radial distance of the satellite from the center of a spherical Earth (Bate, 1971, pp. 11 - 16). The theory used in the NAVSPASUR model assumes only that Earth is symmetrical about the north-south axis. Expressing the potential in spherical harmonics, the Earth's gravitational potential is modeled by

$$U=\frac{\mu}{r}+\frac{\mu}{r}\sum_{n=2}^{\infty}\frac{R_{\oplus}^{n}}{r^{n}}\sum_{q=0}^{n}P_{n}^{q}(\sin\beta)\,[C_{n,q}\cos q\lambda+S_{n,q}\sin q\lambda] \tag{2.2}$$

where $R_{\oplus}$ is the equatorial radius of the Earth, β is the satellite latitude, λ is the longitude, $C_{n,q}$ and $S_{n,q}$ are

coefficients depending on the mass distribution, and P_n^q are the associated Legendre polynomials. These polynomials are defined in terms of the Legendre polynomials P_n:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_n(x) &= \frac{2n-1}{n} x P_{n-1}(x) - \frac{n-1}{n} P_{n-2}(x) \\ P_n^q(x) &= (1-x^2)^{q/2} \frac{d^q}{dx^q} P_n(x) \end{aligned} \tag{2.3}$$

Ignoring any longitudinal variation in the gravitational potential, Equation 2.2 may be simplified to

$$U = \frac{\mu}{r} - \frac{\mu}{r} \sum_{n=1}^{\infty} \frac{R_{\oplus}^n}{r^n} J_n P_n(\sin\beta) \tag{2.4}$$

where $J_n = -C_{n,q}$. The even J_n's account for the oblateness of the Earth, while the odd J_n's account for the Earth's asymmetry about the equatorial axis. (Solomon, 1991, p. 3)

1. Brouwer's Model

Dirk Brouwer developed his theory of artificial satellite motion while under contract by the Air Force Cambridge Research Center and published this theory in the *Astronomical Journal* in 1959. In this article, Brouwer used a different notation for the classical orbital elements from the notation commonly recognized today. This notation is also adopted in the later NAVSPASUR model. In order to conform to the notation of Brouwer's original article and NAVSPASUR model, this paper will also use Brouwer's notation listed in Table 2.1.

Table 2.1 Brouwer's Notation

Brouwer's Notation		Common Notation
a	semi-major axis	a
e	eccentricity	e
I	inclination	i
g	argument of perigee	ω
h	ascending node	Ω
f	true anomaly	ν
l	mean anomaly	M

Brouwer's model considers the zonal harmonics of the Earth's gravitational potential, accounting for the Earth's oblateness and its asymmetry about the equatorial axis as expressed in Equation 2.4. To simplify the potential equation, his model uses only the first four non-zero terms of the series described in Equation 2.3 with $J_1=0$:

$$U=\frac{\mu}{r}+\frac{\mu k_2}{r^3}(1-3\sin^2\beta)-\frac{\mu A_{3.0}}{2r^4}(3\sin\beta-5\sin^3\beta)$$
$$+\frac{\mu k_4}{3r^5}(3-30\sin^2\beta+35\sin^4\beta) \qquad (2.5)$$
$$+\frac{\mu A_{5.0}}{8r^6}(15\sin\beta-70\sin^3\beta+63\sin^5\beta)$$

where

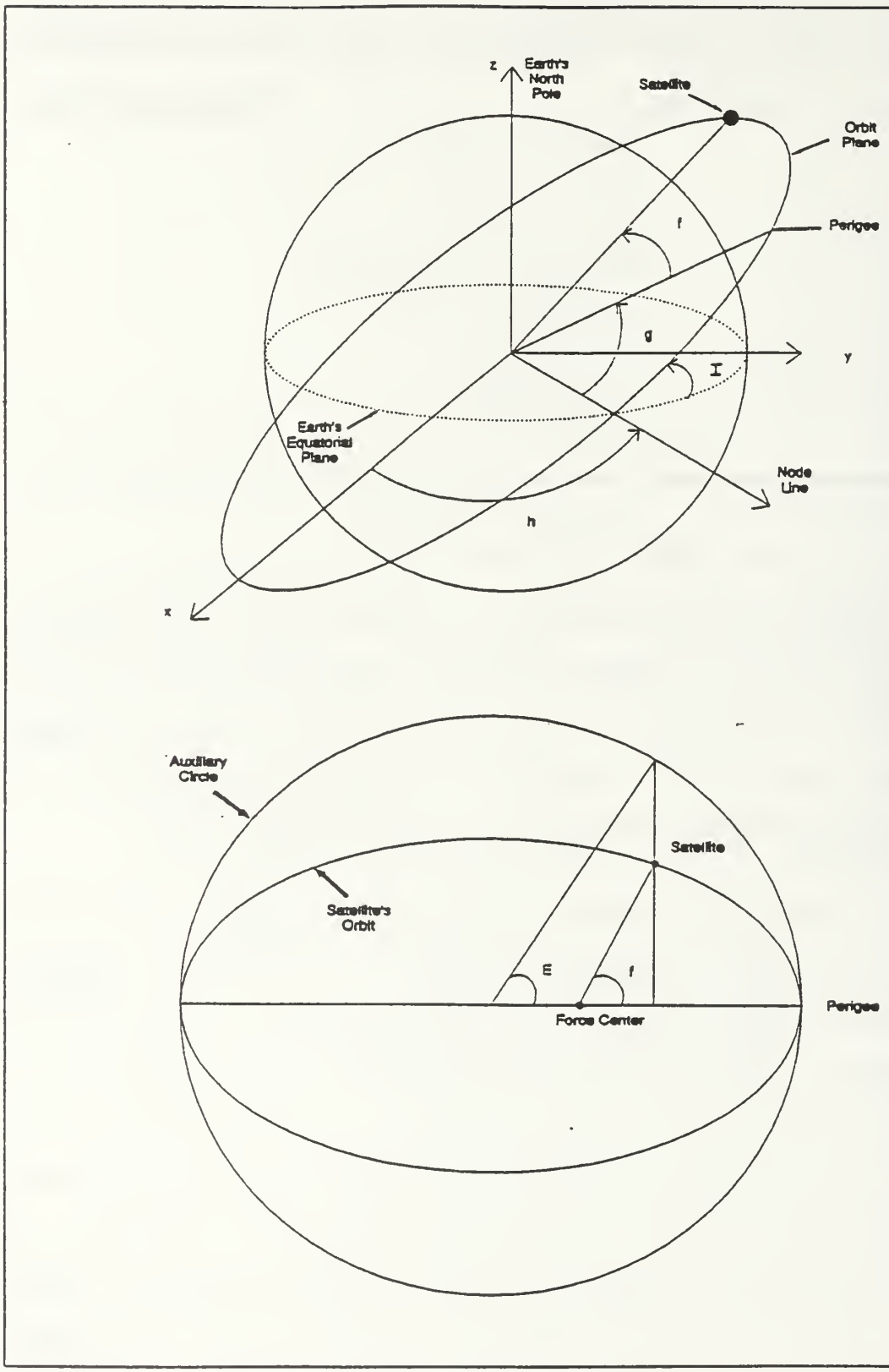

Figure 2.1 Classical Orbital Elements

$$k_2 = \frac{1}{2} J_2 R_\oplus^2$$
$$A_{3.0} = -J_3 R_\oplus^3$$
$$k_4 = -\frac{3}{8} J_4 R_\oplus^4$$
$$A_{5.0} = -J_5 R_\oplus^5$$

This truncation of the series introduces an error of the order $O(k_2^3)$, where $k_2 = .4841605 \cdot 10^{-3} \cdot 0.5\sqrt{5}$. (Brouwer, 1963, p. 393).

In order to derive the equations of motion for the satellite, Brouwer utilized the Hamilton-Jacobi theory of dynamical systems to express the Hamiltonian $F = U - \frac{1}{2}v^2$ in terms of canonically conjugate Delaunay variables. Letting a and e be the osculating semi-major axis and eccentricity, respectively, the Delaunay variables are:

$$\begin{aligned} &L = \sqrt{\mu a} && l \equiv \textit{mean anomaly} \\ &G = \sqrt{\mu a (1-e^2)} && g \equiv \textit{argument of perigee} \\ &H = \sqrt{\mu a (1-e^2)\cos I} && h \equiv \textit{longitude of ascending node} \end{aligned} \qquad (2.6)$$

Using the Delaunay variables in Equations 2.6, the equations of motion become

$$\begin{aligned} \frac{dL}{dt} &= \frac{\partial F}{\partial l}, & \frac{dl}{dt} &= -\frac{\partial F}{\partial L} \\ \frac{dG}{dt} &= \frac{\partial F}{\partial g}, & \frac{dg}{dt} &= -\frac{\partial F}{\partial G} \\ \frac{dh}{dt} &= \frac{\partial F}{\partial h}, & \frac{dh}{dt} &= -\frac{\partial F}{\partial H} \end{aligned} \qquad (2.7)$$

where the hamiltonian is

$$F=\frac{\mu^2}{2L^2}+\frac{\mu^4 k_2}{L^6}\left[\left(-\frac{1}{2}+\frac{3h^2}{2G^2}\right)\frac{a^3}{r^3}+\frac{3}{2}\left(1-\frac{H^2}{G^2}\right)\frac{a^3}{r^3}\cos(2g+2f)\right] \quad (2.8)$$
$$+ \ . \ . \ .$$

In order to express the hamiltonian, F, in terms of only the Delaunay variables, Brouwer used the following Fourier series expansions:

$$\frac{a^3}{r^3}=\frac{L^2}{G^2}+\sum_{j=1}^{\infty}2P_j\cos jl$$
$$\frac{a^3}{r^3}\cos(2g+2f)=\sum_{j=-\infty}^{\infty}Q_j\cos(2g+jl) \quad (2.9)$$

The coefficients P_j, Q_j are power series in the eccentricity, e.

Using two canonical transformations through the choice of suitable determining functions, S and S', Brouwer was able to solve the system of ordinary differential equations listed in Equations 2.7 in terms of the mean elements, a", e", I", l", g", and h". The transformed hamiltonian, F^{**}, depends only on the transformed variables L", G", and H". Replacing F by F^{**} in Equation Set 2.7, Brouwer found that L", G", and H" are constants with respect to time and l", g", and h" are linear functions of time. Consequently, from Equation Set 2.6, it follows that a", e", and I" are constant with respect to time. Additionally, as a consequence of the transformations (see Brouwer, 1959, pp. 379 - 393), Brouwer was able to separate the changes in the orbital elements with time into secular, long period, and short period variations. Including only secular terms up to order $O(k_2^2)$ and periodic terms to order

$O(k_2)$, Brouwer found that the secular variations are a function of only a", e", I", and t; the long period variations are a function of a", e", I", and g"; and the short period variations are a function of all six mean elements. Additionally, a, e, and I do not experience any secular variations to order $O(k_2^2)$ and a does not experience any long period variation to order $O(k_2)$. Using the following constants,

$$
\begin{aligned}
a'' &\equiv \textit{semi-major axis constant} \\
e'' &\equiv \textit{eccentricity constant} \\
I'' &\equiv \textit{inclination constant} \\
n_0 &= \sqrt{\mu a^{-3}} \\
R_\oplus &\equiv \textit{Earth's equatorial radius}
\end{aligned}
$$

and notations,

$$\eta = \sqrt{1-e^2} \qquad \theta = \cos I''$$

$$\gamma_2 = \frac{k_2}{a''^2} \quad \gamma_3 = \frac{A_{3.0}}{a''^3} \quad \gamma_4 = \frac{k_4}{a''^4} \quad \gamma_5 = \frac{A_{5.0}}{a''^5}$$

$$\gamma'_2 = \gamma_2\eta^{-4} \quad \gamma'_3 = \gamma_3\eta^{-6} \quad \gamma'_4 = \gamma_4\eta^{-8} \quad \gamma'_5 = \gamma_5\eta^{-10}$$

Brouwer's formulas for computing the perturbations are:[2]

---

[2]In order to avoid confusion over notation, let δ_*, δ_1, and δ_2 represent the secular, long period, and short period variations, respectively.

Secular terms

$$\delta_s I=\frac{3}{2}\gamma'_2\eta(-1+3\theta^2)$$
$$+\frac{3}{32}\gamma'^2_2\eta[-15+16\eta+25\eta^2+(30-96\eta-90\eta^2)\theta^2+(105+14\eta+25\eta^2)\theta^4]$$
$$+\frac{15}{16}\gamma'_4\eta e''^2(3-30\theta^2+35\theta^4) \qquad (2.10)$$

$$\delta_s g=\frac{3}{2}\gamma'_2(-1+5\theta^2)$$
$$+\frac{3}{32}\gamma'^2_2[-35+24\eta+25\eta^2+(90-192\eta-126\eta^2)\theta^2+(385+360\eta+45\eta^2)\theta^4]$$
$$+\frac{5}{16}\gamma'_4[21-9\eta^2+(-270+126\eta^2)\theta^2+(385-189\eta^2)\theta^4] \qquad (2.11)$$

$$\delta_s h=-3\gamma'_2\theta+\frac{3}{8}\gamma'^2_2[(-5+12\eta+9\eta^2)\theta+(-35-36\eta-5\eta^2)\theta^3] \qquad (2.12)$$
$$+\frac{5}{4}\gamma'_4(5-3\eta^2)(3-7\theta^2)\theta$$

Long period terms

$$\delta_1 e=\{\frac{1}{8}\gamma'_2 e''\eta^2[1-11\theta^2-\frac{40\theta^4}{(1-5\theta^2)}]$$
$$-\frac{5}{12}\frac{\gamma'_4}{\gamma'_2}e''\eta^2[1-3\theta^2-\frac{8\theta^4}{(1-5\theta^2)}]\}\cos 2g''$$
$$+\{\frac{1}{4}\frac{\gamma'_3}{\gamma'_2}\eta^2\sin I'' \qquad (2.13)$$
$$+\frac{5}{64}\frac{\gamma'_5}{\gamma'_2}\eta^2\sin I''(4+3e''^2)[1-9\theta^2-\frac{24\theta^4}{(1-5\theta^2)}]\}\sin g''$$
$$-\frac{35}{385}\frac{\gamma'_5}{\gamma'_2}e''^2\eta^2\sin I''[1-5\theta^2-\frac{16\theta^4}{(1-5\theta^2)}]\sin 3g''$$

$$\delta_1 I=-\frac{e''\delta_1 e}{\eta^2\tan I''} \qquad (2.14)$$

$$\delta_1 I=\eta^3\{\frac{1}{8}\gamma'_2[1-11\theta^2-\frac{40\theta^4}{1-5\theta^2}-\frac{5\gamma'_4}{12\gamma'_2}[1-3\theta^2-\frac{8\theta^4}{1-5\theta^2}]\}\sin 2g''$$
$$-\frac{\eta^3\sin I''}{\gamma'_2 e''}\{\frac{\gamma'_3}{4}+\frac{5\gamma'_5}{64}(4+9e''^2)[1-9\theta^2-\frac{24\theta^4}{1-5\theta^2}]\}\cos g'' \qquad (2.15)$$
$$+\frac{35\gamma'_5}{384\gamma'_2}e''\eta^3\sin I''\{1-5\theta^2-\frac{16\theta^4}{1-5\theta^2}\}\cos 3g''$$

$$\delta_1 g=\{-\frac{\gamma'_2}{16}[(2+e''^2)-11(2+3e''^2)\theta^2-\frac{40(2+5e''^2)\theta^4}{1-5\theta^2}$$
$$-\frac{400e''^2\theta^6}{(1-5\theta^2)^2}]+\frac{5\gamma'_4}{24\gamma'_2}[(2+e''^2)-3(2+3e''^2)\theta^2$$
$$-\frac{8(2+5e''^2)\theta^4}{1-5\theta^2}-\frac{80e''^2\theta^6}{(1-5\theta^2)^2}]\}\sin 2g''$$
$$+\frac{1}{4\gamma'_2}\{\gamma'_3(\frac{\sin I''}{e''}-\frac{e''\theta^2}{\sin I''})$$
$$+\frac{5\gamma'_5}{16}[(\frac{\eta^2\sin I''}{e''}-\frac{e''\theta^2}{\sin I''})(4+3e''^2) \qquad (2.16)$$
$$+e''\sin I''(26+9e''^2)][1-9\theta^2-\frac{24\theta^4}{1-5\theta^2}]$$
$$+\frac{15\gamma'_5}{8}e''\theta^2\sin I''(4+3e''^2)[3+\frac{16\theta^2}{1-5\theta^2}+\frac{40\theta^4}{(1-5\theta^2)^2}]\}\cos g''$$
$$+\frac{35\gamma'_5}{576\gamma'_2}\{-\frac{1}{2}(e''\sin I''(3+2e''^2)-\frac{e''^3\theta^2}{\sin I''})[1-5\theta^2-\frac{16\theta^4}{1-5\theta^2}]$$
$$+e''^3\theta^2\sin I''[5+\frac{32\theta^2}{1-5\theta^2}+\frac{80\theta^4}{(1-5\theta^2)^2}]\}\cos 3g''$$

$$\delta_1 h=e''^2\theta\{-\frac{\gamma'_2}{8}[11+\frac{80\theta^2}{1-5\theta^2}+\frac{200\theta^4}{(1-5\theta^2)^2}]$$
$$+\frac{5\gamma'_4}{12\gamma'_2}[3+\frac{16\theta^2}{1-5\theta^2}+\frac{40\theta^4}{(1-5\theta^2)^2}]\}\sin 2g''$$
$$+\frac{e''\theta}{4\gamma'_2}\{\frac{\gamma'_3}{\sin I''}+\frac{5\gamma'_5}{16\sin I''}(4+3e''^2)[1-9\theta^2-\frac{24\theta^4}{1-5\theta^2}]$$
$$+\frac{15\gamma'_5}{8}\sin I''(4+3e''^2)[3+\frac{16\theta^2}{1-5\theta^2}+\frac{40\theta^4}{(1-5\theta^2)^2}]\}\cos g'' \qquad (2.17)$$
$$-\frac{35\gamma'_5}{576\gamma'_2}e''^3\theta\{\frac{1}{2\sin I''}[1-5\theta^2-\frac{16\theta^4}{1-5\theta^2}]$$
$$+\sin I''[5+\frac{32\theta^2}{1-5\theta^2}+\frac{80\theta^4}{(1-5\theta^2)^2}]\}\cos 3g''$$

Short period terms

$$\delta_2 a = a'' \gamma'^2 [(-1+3\theta^2) (\frac{a''^3}{r'^3} - \frac{1}{\eta^3}) + 3(1-\theta^2) \frac{a''^3}{r'^3} \cos(2g'+2f')] \quad (2.18)$$

$$\delta_2 e = \frac{\eta^2 \gamma_2}{2e''} [(-1+3\theta^2) (\frac{a''^3}{r'^3} - \frac{1}{\eta^3}) + 3(1-\theta^2) (\frac{a''^3}{r'^3} - \frac{1}{\eta^4} \cos(2g'+2f'')] - \frac{\eta^2 \gamma'^2}{2e''} (1-\theta^2) [3e'' \cos(2g'+f') + e'' \cos(2g'+3f')] \quad (2.19)$$

$$\delta_2 I = \frac{\gamma'^2}{2} \theta \sin I'' [3\cos(2g'+2f') + 3e'' \cos(2g'+f') + e'' \cos(2g'+3f')] \quad (2.20)$$

$$\delta_2 l = -\frac{\eta^3 \gamma'^2}{4e''} \{ 2(-1+3\theta^2) (\frac{a''^2}{r'^2} \eta^2 + \frac{a''}{r'} + 1) \sin f' + 3(1-\theta^2) [1 - \frac{a''^2}{r'^2} \eta^2 + \frac{a''}{r'}) \sin(2g'+f') + (\frac{a''^2}{r'^2} \eta^2 + \frac{1}{3}) \sin(2g'+3f')] \} \quad (2.21)$$

$$\delta_2 g = \frac{\eta^2 \gamma'^2}{4e''} \{ 2(-1+3\theta^2) [(\frac{a''^2}{r'^2} \eta^2 + \frac{a''}{r'} + 1) \sin f' + 3(1-\theta^2) [(1 - \frac{a''^2}{r'^2} \eta^2 + \frac{a''}{r'}) \sin(2g'+f') + (\frac{a''^2}{r'^2} \eta^2 + \frac{a''}{r'} + \frac{1}{3}) \sin(2g'+3f')] \} + \frac{\gamma'^2}{4} \{ 6(-1+5\theta^2) (f'-l'+e'' \sin f') + (3-5\theta^2) (3\sin(2g'+2f') + 3e'' \sin(2g'+f') + e'' \sin(2g'+3f') \} \quad (2.22)$$

$$\delta_2 h = -\frac{\gamma'^2}{2}\theta\{6(f'-l'+e''\sin f') - (3\sin(2g'+2f') + 3e''\sin(2g'+f') + e''\sin(2g'+3f')\} \quad (2.23)$$

Using Equations 2.10 through 2.23, Brouwer's algorithm for predicting a satellite's state vector is as follows:

- Begin with mean elements a", e", I", l_0", g_0", and h_0".
- Form mean motion n_0.

$$n_0 = \sqrt{\mu a''^{-3}} \quad (2.24)$$

- Propagate "mean" mean anomaly l", mean argument of perigee g", and mean ascending node h" using Equations 2.10 - 2.12.

$$\begin{aligned} l'' &= l_0'' + n_0 t(1+\delta_s l) \\ g'' &= g_0'' + n_0 t\delta_s g \\ h'' &= h_0'' + n_0 t\delta_s h \end{aligned} \quad (2.25)$$

- Apply long periodic corrections to l", g", and h" using Equations 2.15 - 2.17 and compute the long period variations $\delta_1 e$ and $\delta_1 I$ using Equations 2.13 and 2.14.

$$\begin{aligned} l' &= l'' + \delta_1 l \\ g' &= g'' + \delta_1 g \\ h' &= h'' + \delta_1 h \end{aligned} \quad (2.26)$$

- Solve Kepler's Equation for the eccentric anomaly E', using l' and e" and compute the true anomaly, f' and radius, r'.

$$\begin{aligned} &E' - e''\sin E' = l'' \\ &\tan\frac{f'}{2} = \sqrt{\frac{(1+e'')}{(1-e'')}}\tan\frac{E'}{2} \\ &r' = \frac{a''(1-e^2)}{1+e''\cos f'} \end{aligned} \quad (2.27)$$

- Apply short period variations to l', g', h', a", e", and I" using Equations 2.18 - 2.23.

$$
\begin{aligned}
l &= l' + \delta_2 l \\
g &= g' + \delta_2 g \\
h &= h' + \delta_2 h \\
a &= a'' + \delta_2 a \\
e &= e'' + \delta_1 e + \delta_2 e \\
I &= I'' + \delta_1 I + \delta_2 I
\end{aligned}
\tag{2.28}
$$

- Solve Kepler's Equation for E, using e and l and compute f and r.

$$
\begin{aligned}
E - e\sin E &= l \\
\tan\frac{f}{2} &= \sqrt{\frac{(1+e)}{(1-e)}}\tan\frac{E}{2} \\
r &= \frac{a(1-e^2)}{1+e\cos f}
\end{aligned}
\tag{2.29}
$$

- Compute the position vector in the conventional manner.

$$
\begin{aligned}
x &= r[\cos(g+f)\cos h - \sin(g+f)\sin h\cos I] \\
y &= r[\cos(g+f)\sin h + \sin(g+f)\cos h\cos I] \\
z &= r\sin(g+f)\sin I
\end{aligned}
\tag{2.30}
$$

In addition to accounting for perturbations only due the Earth's oblateness and asymmetry about the equatorial axis, this model has several shortcomings which Brouwer addressed in his article. The first is a singularity at critical inclination ($I_c = \cos^{-1}(1/\sqrt{5}) \approx 63.4°$) for the long period corrections since many of the terms have a divisor of $(5\cos^2 I - 1)$. Second is a singularity for very small eccentricities (near circular orbits). This singularity is due to the appearance of e" as a divisor in the short period terms. Finally, there exists a singularity in some of the elements for very small inclinations (orbits lying in the

equatorial plane). Brouwer suggested that although singularities in some of the elements existed for very small eccentricities and inclinations, no such singularity existed in the coordinates. Hence, the formulas could be modified and expressions obtained for the perturbations in coordinates for these special cases.(Brouwer,1959, p. 393)

2. Lydanne's Modifications

Lydanne's modifications correct for the singularities in Brouwer's model due to the very small eccentricities and inclinations. He presented the suggested modifications in an article in the *Astronomical Journal* in 1963.

As Brouwer suggested in his article, Lydanne and several other investigators believed there existed well-determined expressions for the coordinates of a satellite in the case of either very small eccentricity or inclination, because no singularity actually existed in the coordinates for the small eccentricity or inclination. However, Lydanne encountered difficulty in applying the approach suggested by Brouwer. This approach requires the Taylor series expansion of the coordinates in the element perturbation. Although the first-order terms were regular, the higher order terms were singular. (Lyddane, 1963, p. 555)

Lyddane suggested another approach. By formulating the perturbation theory in terms of Poincare' variables instead of Delaunay variables, the singularities can be

avoided. The Poincare' variables are defined in the terms of the Delaunay variables as follows:

$$\begin{aligned} x_1 &= L & y_1 &= l+g+h \\ x_2 &= \sqrt{2(L-G)}\cos(g+h) & y_2 &= -\sqrt{2(L-G)}\sin(g+h) \\ x_3 &= \sqrt{(G-H)}\cos h & y_3 &= -\sqrt{2(G-H)}\sin h \end{aligned} \tag{2.31}$$

Using a method similar to Brouwer's method of solution, Lyddane presented with some algebraic manipulation the following formulas:

$$\begin{aligned} a &= a'' + \delta a \\ l+g+h &= l''+g''+h''+\delta(l+g+h) \end{aligned} \tag{2.32}$$

$$\begin{aligned} e\cos l &= (e''+\delta e)\cos l'' - e''\delta l \sin l'' \\ e\sin l &= (e''+\delta e)\sin l'' + e''\delta l \cos l'' \end{aligned} \tag{2.33}$$

$$\begin{aligned} \sin\left(\frac{I}{2}\right)\cosh &= \left[\sin\left(\frac{I''}{2}\right)+\cos\left(\frac{I''}{2}\right)\frac{\delta I}{2}\right]\cosh'' \\ &\quad -\sin\left(\frac{I''}{2}\right)\delta h \sinh'' \\ \sin\left(\frac{I}{2}\right)\sinh &= \left[\sin\left(\frac{I''}{2}\right)+\cos\left(\frac{I''}{2}\right)\frac{\delta I}{2}\right]\sinh'' \\ &\quad +\sin\left(\frac{I''}{2}\right)\delta h \cosh'' \end{aligned} \tag{2.34}$$

where

$$\delta = \delta_1 + \delta_2$$

Additionally, Lydanne discovered that the use of l" instead of l' in Equations 2.21 and 2.22 introduces an error of at most order $O(k_2^2)$. Since Brouwer model computes long period terms to only to order $O(k_2)$, Lyddane suggested that the short period corrections may be computed using l" instead of l'

(Lydanne, 1963, p. 557). Lyddane claimed this approach would overcome the e"=0 and I"=0 singularities in the periodic variations. His approach removed the singularity e"=0 for all terms and the singularity at I"=0 for all terms except $\delta_1 I$ (Equation 2.14) (Solomon, 1991, p. 8).

Lyddane's algorithm for propagating an element set is as follows:

- Compute l", h", δe, e"δl, and δI using Brouwer's formulas (Equations 2.10 - 2.23). To avoid a small difference between two large quantities, use the following identities in Equation 2.19 to compute δe.

$$\left(\frac{1}{e''}\right)\left[\left(\frac{a''}{r''}\right)^3-\eta^{-3}\right]=\frac{1}{\eta^6}\left(e''\eta+\frac{e''}{1+\eta}+3\cos f''+3e''\cos^2 f''+e''^2\cos^3 f''\right) \quad (2.35)$$

$$\left(\frac{1}{e''}\right)\left[\left(\frac{a''}{r''}\right)^3-\eta^{-4}\right]=\frac{1}{\eta^6}\left(e''+3\cos f''+3e''\cos^2 f''+e''^2\cos^3 f''\right)$$

- Compute a and 1+g+h using Equation Set 2.32. To reduce amount of error introduced by finite precision, combine Equations 2.15 - 2.17 for $\delta_1(1+g+h)$ and Equations 2.21 - 2.23 for $\delta_2(1+g+h)$.

- Compute $\sin(\frac{1}{2}I'')\,\delta h$.

$$\sin\left(\frac{I''}{2}\right)\delta h=\frac{\sin I''\delta h}{2\cos\left(\frac{I''}{2}\right)} \quad (2.36)$$

- Solve for e, I, l, and h using Equation Sets 2.33 and 2.34.

- Compute coordinates and velocity components in the usual manner.

$$\hat{f}=\begin{bmatrix}\cos h\cos(g+f)-\sin h\sin(g+f)\cos I\\ \sin h\cos(g+f)+\cos h\sin(g+f)\cos I\\ \sin(g+f)\sin I\end{bmatrix}$$

$$\hat{\vartheta}=\begin{bmatrix}-\cos h\sin(g+f)-\sin h\cos(g+f)\cos I\\ -\sin h\sin(g+f)+\cos h\cos(g+f)\cos I\\ \cos(g+f)\sin I\end{bmatrix} \qquad (2.37)$$

$$\vec{r}=r\hat{f}$$

$$\vec{v}=\frac{[(e\sin f)\hat{f}+(1+e\cos f)\hat{\vartheta}]}{\eta\sqrt{a}}$$

3. NAVSPASUR Modifications

a. Atmospheric Drag

Brouwer's model considers only the perturbations due to zonal harmonics (Earth's oblateness and asymmetry about the equatorial axis). For near-earth orbits, the magnitude of the perturbative acceleration due to atmospheric drag can be of the same order as the magnitude of the perturbative acceleration due of the earth's oblateness (Knowles, 1992, p. 226). Figure 2.2 shows the relative orders of magnitude of the perturbative accelerations at various altitudes above the Earth. Fluctuations in the density and relative velocity of the atmosphere to the satellite make the perturbative acceleration due to atmospheric drag difficult to model. To compensate for the drag perturbation, the NAVSPASUR model utilizes a simplified sub-model for drag (Solomon, 1991, pp. 9 - 10). Atmospheric drag is modeled by time derivatives of the "mean" mean anomaly, l", using two parameters M_2 and M_3:

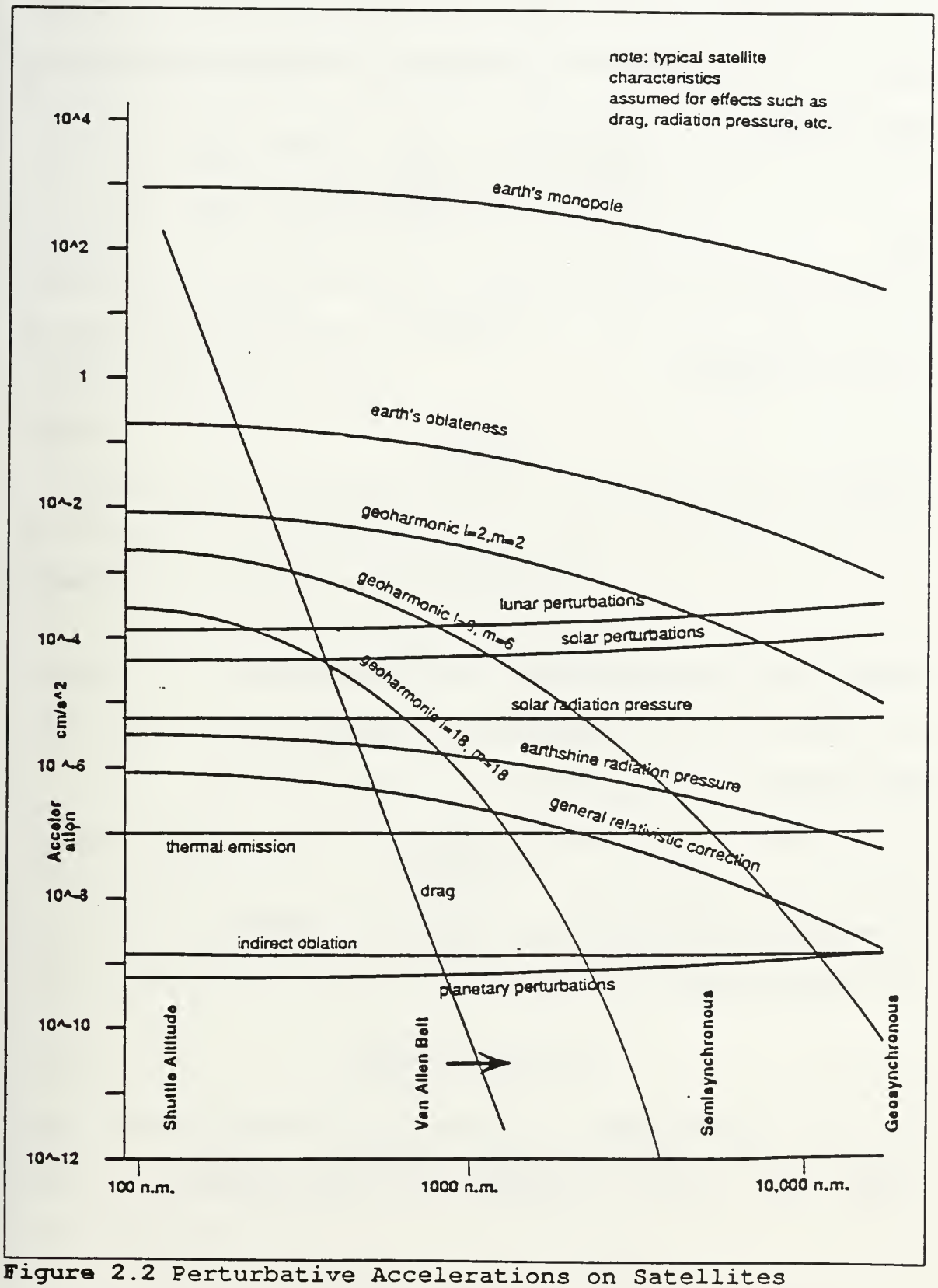

Figure 2.2 Perturbative Accelerations on Satellites

$$l''=l''_0+mt+M_2t^2+M_3t^3 \tag{2.38}$$

where t is the elapsed time from epoch. Using the following relationships,

$$m=n_0(1+\delta_s l)\approx n_0$$

and

$$a''^3=n_0^{-2}$$

it can be assumed

$$a''^3\approx m^{-2} \tag{2.39}$$

Differentiating Equation 2.39 and working to first order,

$$\dot{a}=-\frac{2}{3}\frac{\dot{m}}{m^{\frac{5}{3}}} \tag{2.40}$$

Using $\dot{m}\approx 2M_2$ from Equation 2.38 and substituting into Equation 2.40 yields following model for the drag effect on the semi-major axis:

$$\dot{a}=-\frac{4a''}{3m}M_2 \tag{2.41}$$

Assuming a spherically symmetric atmosphere with an exponential density variation

$$\rho(r)=\rho_0\exp(-\frac{r-r_0}{H}) \tag{2.42}$$

where ρ_0 is the density at radius $r=r_0$ and H is the scale height, the rates of change in the semi-axis, a, and

eccentricity, e, with respect to the eccentric anomaly, E, may be expressed as (Danby, 1988, pp. 330 - 331):

$$\frac{da}{dE}=-a^2\delta\rho_0\exp\left(\frac{Aae}{H}\cos E\right)\sqrt{\frac{1+e\cos E}{1-e\cos E}}(1+e\cos E)$$
$$\frac{de}{dE}=-a\eta\delta\rho_0\exp\left(\frac{Aae}{H}\cos E\right)\sqrt{\frac{1+e\cos E}{1-e\cos E}}\cos E \qquad (2.43)$$

where δ and A are constants. Estimating the average change in a and e by integrating Equation Set 2.43 over one orbit to lowest order of e yields:

$$\frac{\Delta e}{\Delta a}=\frac{e\eta^2}{a}\sigma \qquad (2.44)$$

Assuming σ=1 and substituting Equation 2.41 into 2.44, the model for the decay in eccentricity is:

$$\dot{e}=\frac{e''\eta^2\dot{a}}{a''}=-\frac{4e''\eta^2}{3m}M_2 \qquad (2.45)$$

b. Near Critical Inclination

Neither Brouwer nor Lyddane were able to correct for the singularity in the long period terms at the critical inclination ($I_c=\cos^{-1}(1/\sqrt{5})\approx 63.4°$). To prevent an overflow error in the subroutine **PPT2**, the factor $(1-5\cos^2 I'')^{-1}$ is approximated by the function

$$T2=\frac{1-\exp(-100x^2)}{x} \qquad (2.46)$$

where $x=1-5\cos^2 I''$. However, with a small value of x, T2 cannot be computed directly. By using a product expansion of Equation 2.46,

$$T2=\frac{1}{x}(1-\exp(-\beta x^2))\prod_{n=0}^{10}(1+\exp(-2^n\beta x^2)) \qquad (2.47)$$

where $\beta=100/2^{11}$. To remove the factor of x from the denominator, the first factor is approximated by

$$\frac{(1-\exp(-\beta x^2))}{x}=\beta x\sum_{n=0}^{12}(-1)^n\frac{\beta^n x^{2n}}{(n+1)!} \qquad (2.48)$$

Figure 2.3 shows a comparison of T2 using Equations 2.47 and 2.48 and 1/x in vicinity of the critical inclination. (Solomon, 1991, pp. 10 - 11)

C. PPT2

PPT2 is the Fortran subroutine which implements Brouwer's model with Lyddane's and NAVSPASUR's modifications. **PPT2** completes several satellite propagation tasks.

- Predicts a satellite state vector at future time.
- Computes partial derivatives of the position vector with respect to the orbital elements (used in Method of Differential Correction to modify set of stored elements in light of current observations).
- Predicts the time and state vector of satellite for a given true anomaly.

PPT2 is composed of sections which accomplish the tasks named above. The sections are delineated by conditional breakpoints. Control over which section to execute is handled by a set of control variables. Data is passed to the subroutine through three control variables and four *Common* blocks. A complete description of the input/output of **PPT2** is

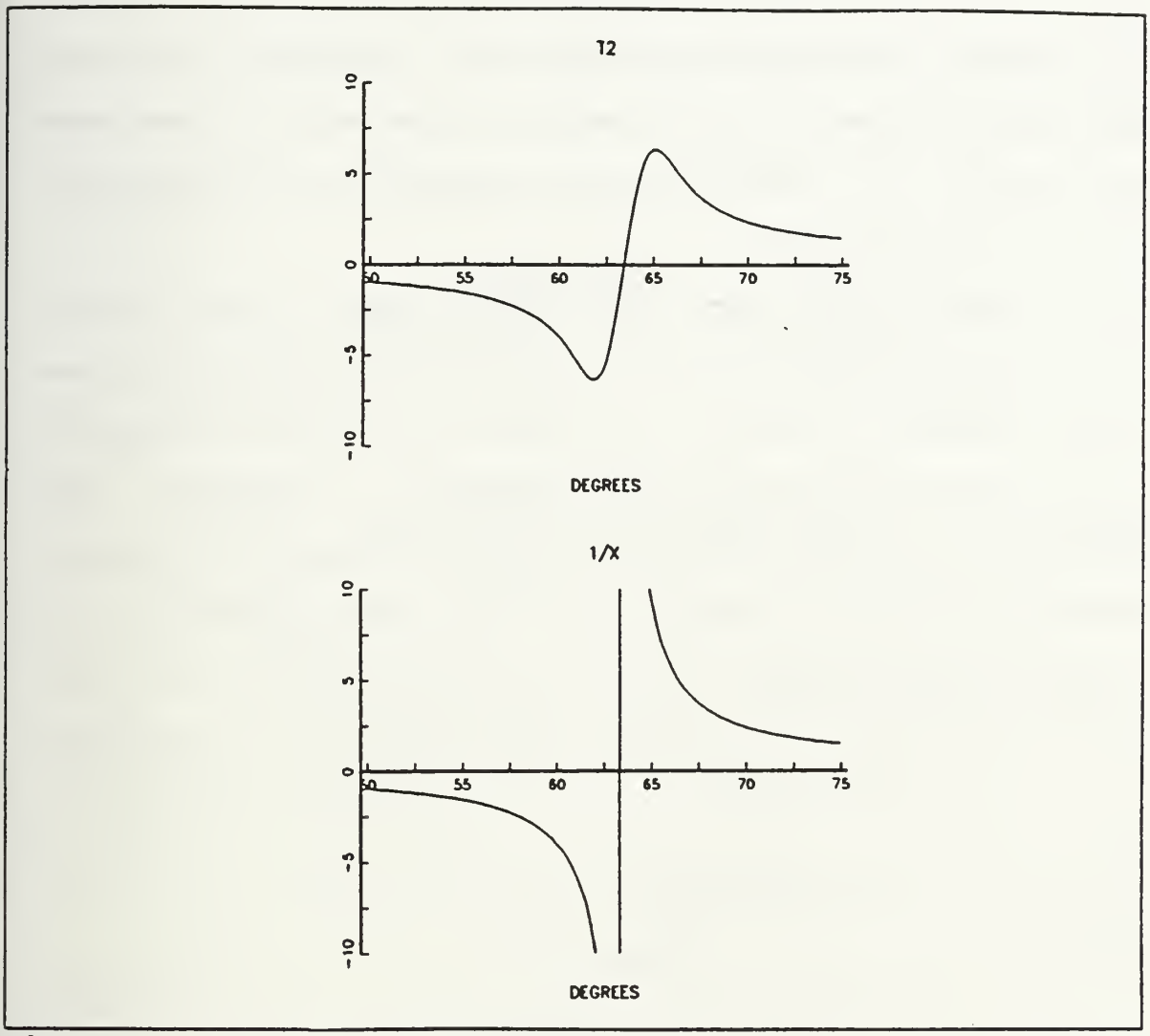

Figure 2.3 Near Critical Inclination

contained in Appendix A.

NAVSPASUR represents each tracked object by a set of stored elements. The stored elements are the mean orbital elements plus two drag parameters, M_2 and M_3. The stored orbital elements are the same as the classical orbital elements used in Brouwer's model with two exceptions. The mean semi-major axis, a", is replaced by the variable, m, the mean "mean" motion.

$$m=(1+\delta_s 1)\,n_0=(1+\delta_s 1)\,a''^{-2/3} \qquad (2.49)$$

The other exception is that the cosine of the mean inclination, cos I", is stored instead of the mean inclination, I". Thus the stored elements are: l", m, M_2, M_3, e", g", h", and cos I".

For NAVSPASUR to use the formulas from Brouwer's model, the mean semi-major axis, a", must be determined from the mean "mean" motion, m. Since a" is implied in the $\delta_s 1$ (Equation 2.10), no direct solution is possible. Therefore, **PPT2** recovers a" using an initial approximation for a" and solving Equation 2.48 for a" iteratively. The initial guess for a" is $a_0''=m^{-\frac{2}{3}}$. Then, for i=1,...,5

$$\begin{aligned}
\gamma'_{2i} &= \frac{k_2}{a''^{\,2}_{i-1}\eta^4} \\
\gamma'_{4i} &= \frac{k_4}{a''^{\,4}_{i-1}\eta^8} \\
(\delta_s 1)_i &= \frac{3}{2}\gamma'_{2i}\eta(-1+3\theta^2) \\
&\quad + \frac{3}{32}\gamma'_{2i}\eta[-15+16\eta+25\eta^2 \\
&\quad + (30-96\eta-90\eta^2)\theta^2+(105+14\eta+25\eta^2)\theta^4] \\
&\quad + \frac{15}{16}\gamma'_{4i}\eta e''^2(3-30\theta^2+35\theta^4) \\
a''_i &= \left(\frac{1+\delta_s 1_i}{m}\right)^{2/3}
\end{aligned} \qquad (2.50)$$

and the mean semi-major axis, a''_0, is taken to be equal to a_5'' (Solomon, 1991, p. 15).

Another difference between the model and actual computations by **PPT2** is the computation of the secular

corrections for the mean argument of perigee, g", and mean ascending node, h". The secular corrections for g" and h" are computed in terms of mean anomaly, l", instead of time using $\delta_s g$ and $\delta_s h$ from Equations 2.11 and 2.12.

$$\frac{dg''}{dl''} = \frac{\frac{dg''}{dt}}{\frac{dl''}{dt}} = \frac{\frac{dg''}{dt}}{m} = \frac{n_0 \delta_s g}{m} = \frac{\delta_s g}{m a''^{3/2}} \qquad (2.51)$$

$$\frac{dh''}{dl''} = \frac{\frac{dh''}{dt}}{\frac{dl''}{dt}} = \frac{\frac{dh''}{dt}}{m} = \frac{n_0 \delta_s h}{m} = \frac{\delta_s h}{m a''^{3/2}} \qquad (2.52)$$

Once the secular corrections to the "mean" mean anomaly are computed via Equation 2.38, Equations 2.51 and 2.52 are used to correct g" and h".

$$\begin{aligned} g'' &= g_0'' + \frac{dg}{dl}\Delta l \\ h'' &= h_0'' + \frac{dh}{dl}\Delta l \end{aligned} \qquad (2.53)$$

where

$$\Delta l = mt + M_2 t^2 + M_3 t^3$$

With all of its conditional breakpoints, **PPT2** completes only those tasks required by the user. Prior to completing any of the fore-mentioned tasks, the user is required to make an initial call to subroutine **PPT2** to recover a" and compute the secular corrections. During this initial call, many variables are set which will be used in subsequent calls,

increasing efficiency. Therefore, at least two calls to **PPT2** are required to complete any of its tasks.[3]

Because the main objective of this thesis is the parallelization of the satellite state vector prediction task of **PPT2**, only this algorithm will be presented. For a complete description of the other tasks see (Solomon, 1991).[4] The algorithm implementing the NAVSPASUR model is as follows:

- Begin with the stored mean elements plus the drag correction terms (l_0", m, M_2, M_3, e_0", g_0", h_0", and cos I")
- Compute T2 using Equations 2.47 and 2.48.
- Recover mean semi-major axis, a_0", from the mean motion, m, by iteration using Equation Set 2.50.
- Compute the following dimensionless quantities:

$$\gamma_2=\frac{k_2}{a''^2} \quad \gamma_3=\frac{A_{3.0}}{a''^3} \quad \gamma_4=\frac{k_4}{a''^4} \quad \gamma_5=\frac{A_{5.0}}{a''^5} \qquad (2.54)$$
$$\gamma'_2=\gamma_2\eta^{-4} \quad \gamma'_3=\gamma_3\eta^{-6} \quad \gamma'_4=\gamma_4\eta^{-8} \quad \gamma'_5=\gamma_5\eta^{-10}$$

- Compute drag corrections for the semi-major axis, a", and eccentricity, e", using Equations 2.41 and 2.45.
- Using Equation 2.38 and Equation Set 2.53, propagate the mean anomaly, l", argument of perigee, g", and the ascending node, h", considering only the secular corrections. (h" may be optionally corrected for the Earth's rotation using Equation 2.56, where ω is the Earth's angular velocity and T_e is the time at which the direction of the Greenwich meridian and equinox direction coincides.

---

[3]For a complete description of calling options of PPT2, see (Solomon, 1991, pp. 11 - 24).

[4]A complete listing of subroutine PPT2 is contained in (Solomon, 1991, pp. 39 - 55).

$$
\begin{aligned}
l'' &= l_0'' + \Delta l \\
g'' &= g_0'' + \frac{dg}{dl}\Delta l \\
h'' &= h_0'' + \frac{dh}{dl}\Delta l
\end{aligned}
\tag{2.55}
$$

$$
h'' = h'' - \omega(t - T_c) \quad [optional] \tag{2.56}
$$

- *Propagate the eccentricity, e", and the semi-major axis, a".*

$$
\begin{aligned}
e'' &= \min[\max[0, e_0'' + \dot{e}t], .99999] \\
a'' &= \max[1, a_0'' + \dot{a}t]
\end{aligned}
\tag{2.57}
$$

- Solve Kepler's Equation using Steffensen's Method and use this result to compute cosine and sine of the true anomaly and radius.

$$
\begin{aligned}
E'' &= l'' + e''\sin E'' \\
\cos f'' &= \frac{\cos E'' - e''}{1 - e''\cos E''} \\
\sin f'' &= \frac{\sqrt{1 - e''^2}\,\sin E''}{1 - e''\cos E''} \\
r'' &= \frac{a''\eta^2}{1 + e''\cos f''}
\end{aligned}
\tag{2.58}
$$

- Using Equations 2.13-2.17, compute long period correction terms for e, l, h, and I in the following forms replacing g′ and $(1-5\theta^2)^{-1}$ with g" and T2, respectively:

$$
\begin{aligned}
\delta_1 e &= VLE1\cos 2g'' + VLE2\sin g'' + VLE3\sin 3g'' \\
e''\delta_1 l &= \eta(VLE1\sin 2g'' - VLL2\cos g'' - VLE3\cos 3g'') \\
\sin I''\delta_1 h &= VLH1I\sin 2g'' + VLH2I\cos g'' + VLH3I\cos 3g'' \\
\delta_1 I &= -\frac{e''\delta_1 e}{\eta^2 \tan I''}
\end{aligned}
\tag{2.59}
$$

where

$$VLE1=e''\eta^2\left(\frac{1}{8}\gamma'_2[1-11\theta^2-40\theta^4\cdot T2]-\frac{5\gamma'_4}{12\gamma'_2}[1-3\theta^2-8\theta^4\cdot T2]\right)$$

$$VLE2=\frac{\eta^2 sinI''}{4\gamma'_2}\left(\gamma'_3+\frac{5\gamma'_5}{16}(4+3e''^2)[1-9\theta^2-24\theta^4\cdot T2]\right)$$

$$VLE3=-\frac{35\gamma'_5}{384\gamma'_2}e''^2\eta^2 sinI''[1-5\theta^2-16\theta^4\cdot T2]$$

$$VLL2=VLE2+\frac{15\gamma'_5}{32\gamma'_2}e''^2\eta^2 sinI''[1-9\theta^2-24\theta^4\cdot T2]$$

$$VLH1I=e''^2\theta sinI''\left(-\frac{\gamma'_2}{8}[11+80\theta^2\cdot T2+200\theta^4\cdot T2^2]\right.$$
$$\left.+\frac{5\gamma'_4}{12\gamma'_2}[3+16\theta^2\cdot T2+40\theta^4\cdot T2^2]\right)$$

$$VLH2I=\frac{e''\theta}{4\gamma'_2}\left\{\gamma'_3+\frac{5\gamma'_5}{16}(4+3e''^2)[1-9\theta^2-24\theta^4\cdot T2]\right.$$
$$\left.+\frac{15\gamma'_5 sin^2 I''}{8}(4+3e''^2)[3+16\theta^2\cdot T2+40\theta^4\cdot T2^2]\right\}$$

$$VLH3I=-\frac{35\gamma'_5}{576\gamma'_2}e''^3\theta\left\{\frac{1}{2}[1-5\theta^2-16\theta^4\cdot T2]\right.$$
$$\left.+sin^2 I''[5+32\theta^2\cdot T2+80\theta^4\cdot T2^2]\right\}$$

- Representing the quantity (1+g+h) by z, compute $\delta_1 z=\delta_1 1+\delta_1 g+\delta_1 h$ by combining individual parts prior to computing.

$$\delta_1 z=VLS1 sin2g''+VLS2 cosg''+VLS3 cos3g'' \qquad (2.60)$$

where

$$VLS1=\frac{(\eta^3-1)}{8}\left[\gamma'_2(1-11\theta^2-40\theta^4\cdot T2)-\frac{10\gamma'_4}{3\gamma'_2}(1-3\theta^2-8\theta^4\cdot T2)\right]$$
$$+\frac{e''^2\theta}{8}\left[\gamma'_2(11+80\theta^2\cdot T2+200\theta^4\cdot T2^2)-\gamma'_3(3+16\theta^2\cdot T2+40\theta^4\cdot T2)\right]$$
$$+25e''^2\theta^6\cdot T2^4\left(\gamma'_2-\frac{\gamma'_3}{5}\right)-\frac{e''^2}{16}\left[\gamma'_2(1-33\theta^2-200\theta^4\cdot T2)\right.$$
$$\left.-\gamma'_3(1-9\theta^2-40\theta^4\cdot T2)\right]$$

$$VLS2=e''\sin I''\{\frac{1}{4\gamma'_2}[(\eta+\frac{1}{1+\eta})+\frac{\theta}{1+\theta}][\gamma'_3+\frac{5\gamma'_5}{16}(4+3e''^2)(1-9\theta^2-24\theta^4\cdot T2)]$$
$$+\frac{10\gamma'_5}{64\gamma'_2}[3(e''^2-\eta^3+11][1-9\theta^2-24\theta^4\cdot T2]$$
$$+\frac{15\gamma'_5}{32\gamma'_2}\theta(1-\theta)[(4+3e''^2)(3+16\theta^2\cdot T2+40\theta^4\cdot T2^2)]\}$$

$$VLS3=e''sinI''\{\frac{35\gamma'_5}{1152\gamma'_2}[3\eta^2-3-(2+\frac{\theta}{1+\theta})e''^2][1-5\theta^2-16\theta^4\cdot T2]$$
$$-\frac{35\gamma'_5}{576\gamma'_2}[e''^2\theta(1-\theta)(5+40\theta^2\cdot T2+80\theta^4\cdot T2^2)]\}$$

- Using Equation 2.19 and Equation Set 2.35, compute the short period correction term for e, replacing g′ and f′ with g" and f", respectively. Then combine long and short period corrections to form δe.

$$\delta_2 e=\frac{\gamma'^2}{2}\{(-1+3\theta^2)[e''^2\cos^3 f''+3e''\cos^2 f''$$
$$+3\cos f''+e''(\eta+\frac{1}{1+\eta})] \qquad (2.61)$$
$$+3(1-\theta^2)[e''^2\cos^3 f''+3e''\cos^2 f''$$
$$+3\cos f''+e''\cos(2g''+2f'')]$$
$$-\eta^2(1-\theta^2)[3e''\cos(2g''+f'')+e''\cos(2g''+3f'')\}$$

$$\delta e=\delta_1 e+\delta_2 e \qquad (2.62)$$

- Using Equations 2.20 and 2.21, compute the short period correction terms for I and l in the following form replacing g′ and f′ with g" and f", respectively. Then combine respective long and short period terms to form δI and eδl.

$$\delta_2 I=\frac{\gamma'^2}{2}\theta \sin I''\,[3\cos(2g''+2f'')+3e''\cos(2g''+f'')+e''\cos(2g''+3f'')]$$

$$\begin{aligned} e''\delta_2 l=-\frac{\eta^3\gamma'^2}{4}\{&2(-1+3\theta^2)\\ &\times[\frac{(1+e''\cos f'')(2+e''\cos f'')}{\eta}+1]\sin f''\\ &+3(1-\theta^2)\\ &\times[(1-\frac{(1+e''\cos f'')(2+e''\cos f'')}{\eta})\sin(2g''+f'')\\ &+(\frac{(1+e''\cos f'')(2+e''\cos f'')}{\eta}+\frac{1}{3})\\ &\times\sin(2g''+3f'')]\} \end{aligned} \qquad (2.63)$$

$$\begin{aligned} \delta I&=\delta_1 I+\delta_2 I\\ e\delta l&=e\delta_1 l+e\delta_2 l \end{aligned} \qquad (2.64)$$

- Using equation 2.23, compute short period correction for h in the following form.

$$\begin{aligned} \sin I''\delta_2 h=-\frac{\gamma'^2}{2}\sin I''\theta[&6(f''-l''e''\sin f'')\\ &-3\sin(2g''+2f'')+3e''\sin(2g''+f'')\\ &+e''\sin(2g''+3f'')] \end{aligned} \qquad (2.65)$$

- Using Equation 2.36 and relationship $\delta h=\delta_1 h+\delta_2 h$, compute $\sin(\frac{1}{2}I'')\delta h$.

$$\sin(\frac{I''}{2})\delta h=\frac{\sin I''(\delta_1 h+\delta_2 h)}{2\cos(\frac{I}{2})} \qquad (2.66)$$

- Combine terms of $\delta_2 z=\delta_2 l+\delta_2 g+\delta_2 h$, replacing g', l', and f' with g", l", f", respectively. Then compute z.

$$\begin{aligned} \delta_2 z=&-e''^2\delta_2 l\frac{(\eta+\frac{1}{1+\eta}-1)}{\eta^3}\\ &-\frac{\gamma'^2}{4}[6(1-2\theta-5\theta^2)(f''-e''\sin f''-l'')\\ &-(3+2\theta-5\theta^2)(3\sin(2g''+2f'')\\ &+3e''\sin(2g''+2f'')+e''\sin(2g''+3f'')] \end{aligned} \qquad (2.67)$$

$$z = l'' + g'' + h'' + \delta_1 z + \delta_2 z \tag{2.68}$$

- Compute a using Equation 2.18.

$$a = a'' + \delta_2 a \tag{2.69}$$

- Solve Equation Sets 2.33 and 2.34 explicitly for e, l, I, and h.

$$\begin{aligned} e &= \sqrt{(\delta e)^2 + (e\delta l)^2} \\ l &= \tan^{-1}\left(\frac{\delta e \sin l'' + e\delta l \cos l''}{\delta e \cos l'' - e\delta l \sin l''}\right) \\ \cos I &= 1 - 2\left[(\delta I)^2 + \left(\sin\left(\frac{I}{2}\right)\delta h\right)^2\right] \\ h &= \tan^{-1}\left(\frac{\delta I \sin h'' + \sin\left(\frac{I}{2}\right)\delta h \cosh''}{\delta I \cosh'' - \sin\left(\frac{I}{2}\right)\delta h \sinh''}\right) \end{aligned} \tag{2.70}$$

- Compute g.

$$g = z - l - h \tag{2.71}$$

- Solve Kepler's Equation again using Steffensen's Method and compute cos f, sin f, and r.

$$\begin{aligned} E &= l + e\sin E \\ \cos f &= \frac{\cos E - e}{1 - e\cos E} \\ \sin f &= \frac{\sqrt{1 - e^2}\,\sin E}{1 - e\cos E} \\ r &= \frac{a\eta^2}{1 + e\cos f} \end{aligned} \tag{2.72}$$

- Compute the satellite state vector using Equation set 2.37.

III. PARALLEL COMPUTING

A. OVERVIEW

1. Definition

The complexity of scientific computing today demands faster computers. Greater detailed models require a substantial amount of computation. Faster computers are needed to provide the results of the computation in a timely manner. In response to this demand, computer engineers have taken two approaches to achieve faster performance.

The first approach is to increase the speed of the circuitry. Although great advances have been made in increasing the speed of computer circuitry, this increase in speed is bounded by the speed of light. Additionally, the specific design and manufacture requirements for further increases in speed are quite costly.

The second approach, parallel computing, provides an alternate means to achieve faster computer performance using affordable circuitry design. Many articles and books have been written describing the methods to exploit this approach. The terms *parallel computing* and *parallel processing* seem to be used interchangeably in these texts. For the purpose of this thesis, *parallel computing* and *parallel processing* are assumed to be synonymous. In this emerging field, there

exists slight differences in how to define parallel computing. One definition which best encompasses the breadth of the field may be taken from (Hwang, 1984, p. 6):

> *Parallel processing* is an efficient form of information processing which emphasizes the exploitation of concurrent events in the computing process. Concurrency implies parallelism, simultaneity, and pipelining. Parallel events may occur in multiple resources during the same time interval; simultaneous events may occur at the same time instant; and pipelined events may occur in overlapped time spans. These concurrent events are attainable in a computer system at various processing levels.

In his book, Hwang describes the processing levels to be: the program level, the task level, the inter-instruction level, and the intra-instruction level. The program level involves executing multiple programs by means of multiprogramming, time sharing, and multiprocessing. This level is concerned with the design of parallel processing systems which is beyond the scope of this thesis. Therefore, for the purposes of this paper, the definition of *parallel computing* is defined as the efficient form of information processing emphasizing the concurrent computations and manipulation of data to solve a single problem.

2. Classification of Parallel Computers

a. Type Classifications

Implicit in the definition of parallel computing are three methods to achieve parallelism. The three methods are temporal parallelism, spatial parallelism, and asynchronous parallelism (Hwang, 1984, p. 20). These methods

offer a manner to classify the various types of parallel computers.

The first type is a pipeline computer. *Pipeline computers* perform overlapped computations to exploit temporal parallelism. Computations are divided into a number of stages or segments with the output of one segment being the input of another. Analogous to a factory assembly line, if each segment works at same speed, the work rate of the pipeline is the sum of work rates of the segments. The maximum work rate is achieved once the pipeline is full. An example of a pipeline computer is the Cray-1.

The second type is an array processor. *Array processors* use multiple synchronized processing elements to achieve spatial parallelism. Each processing element performs simultaneously identical operations on different data. An example of an array processor is the Connection Machine.

The third type is a multiprocessor. *Multiprocessors* may achieve asynchronous parallelism through a set of interactive processors (nodes). These processors are capable of performing independent operations, but share resources such as memory. An example of a multiprocessor is the Cm* of Carnegie-Mellon University.

The final type is a refinement of the multiprocessor, the multicomputer. *Multicomputers*, like multiprocessors, achieve asynchronous parallelism through a set of interactive processors. But these processors each have

their own local memory. An example of a multicomputer is the INTEL iPSC hypercube. Because each processor has its own memory and may perform independent operations, multicomputers offer the user an added degree of freedom in programming. However, interaction between the processors (nodes) may require synchronization to be explicitly programmed in the multicomputer code.

The four type classifications are not necessarily mutually exclusive. Many commercially available array processors, multiprocessors, and multicomputers employ pipeline processors to complete operations such as vector processing.

b. Architectural Classifications

Parallel computers may also be classified according to their architecture. One scheme for classifying digital computers was introduced by Michael J. Flynn in 1966. He introduced a scheme to classify computers into four categories based on the multiplicity of instruction and data streams. An *instruction stream* is a sequence of instructions to be executed by the computer. Likewise, a *data stream* is a sequence of data used by the computer. Flynn's four categories are (Flynn, 1966):

1. Single instruction stream, single data stream (SISD). Most serial computers fall in the SISD category. Although instructions are completed sequentially, this category includes overlapping instructions (pipelining). Therefore, pure pipeline processors also belong to this category.

2. Single instruction stream, multiple data stream (SIMD). Array processors fall into this category. The array processor receives a single set of instructions, but each element receives and manipulates its own set of data.

3. Multiple instruction stream, single data stream (MISD). No current computers fall into this category. This architecture has been challenged as impractical by some computer designers (Hwang, 1984, p. 34).

4. Multiple instruction stream, multiple data stream (MIMD). Most multiprocessors and multicomputers fall into this category. The INTEL iPSC is a MIMD machine.

c. Topological Classifications

Another classifying scheme for parallel computers is by the topology of the inter-processor connections. These connections are the means through which communication between individual processors is conducted. This classifying scheme applies only to array processors, multiprocessors, and multicomputers. Some of the general topologies are the mesh, the pyramid, the butterfly, and the hypercube. Figures 3.1 and 3.2 show examples of the mesh and hypercube topologies. The topology may also be customized to meet specific computing needs. For a more comprehensive discussion of the various topologies see (Quinn, 1987, pp. 25 - 30).

3. Measurements of Performance

With faster computation speed being the ultimate objective, certain measures are needed to determine the effectiveness of parallel computing versus serial computing to achieve this objective. Computation speed depends on many factors that include the computer hardware design, the

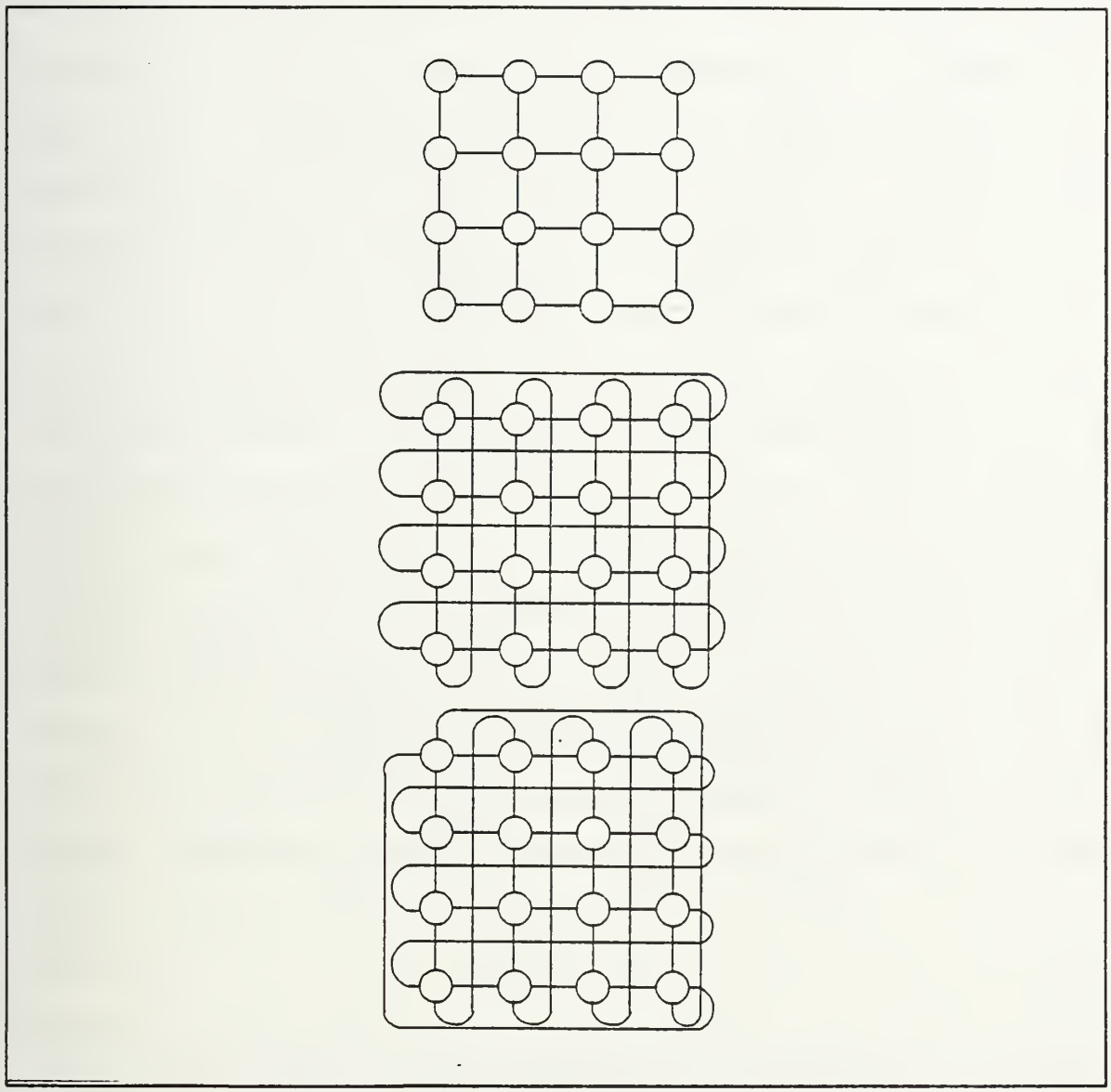

Figure 3.1 Two-dimensional meshes

technical specifications of its components, and the algorithm or method of solution used to complete the computations. Two common measures of effectiveness, accounting for both the hardware and the algorithm, are *speedup* and *efficiency*.

Speedup, S_p, refers to the ratio between the time taken to execute a set of computations serially, T_s, and the

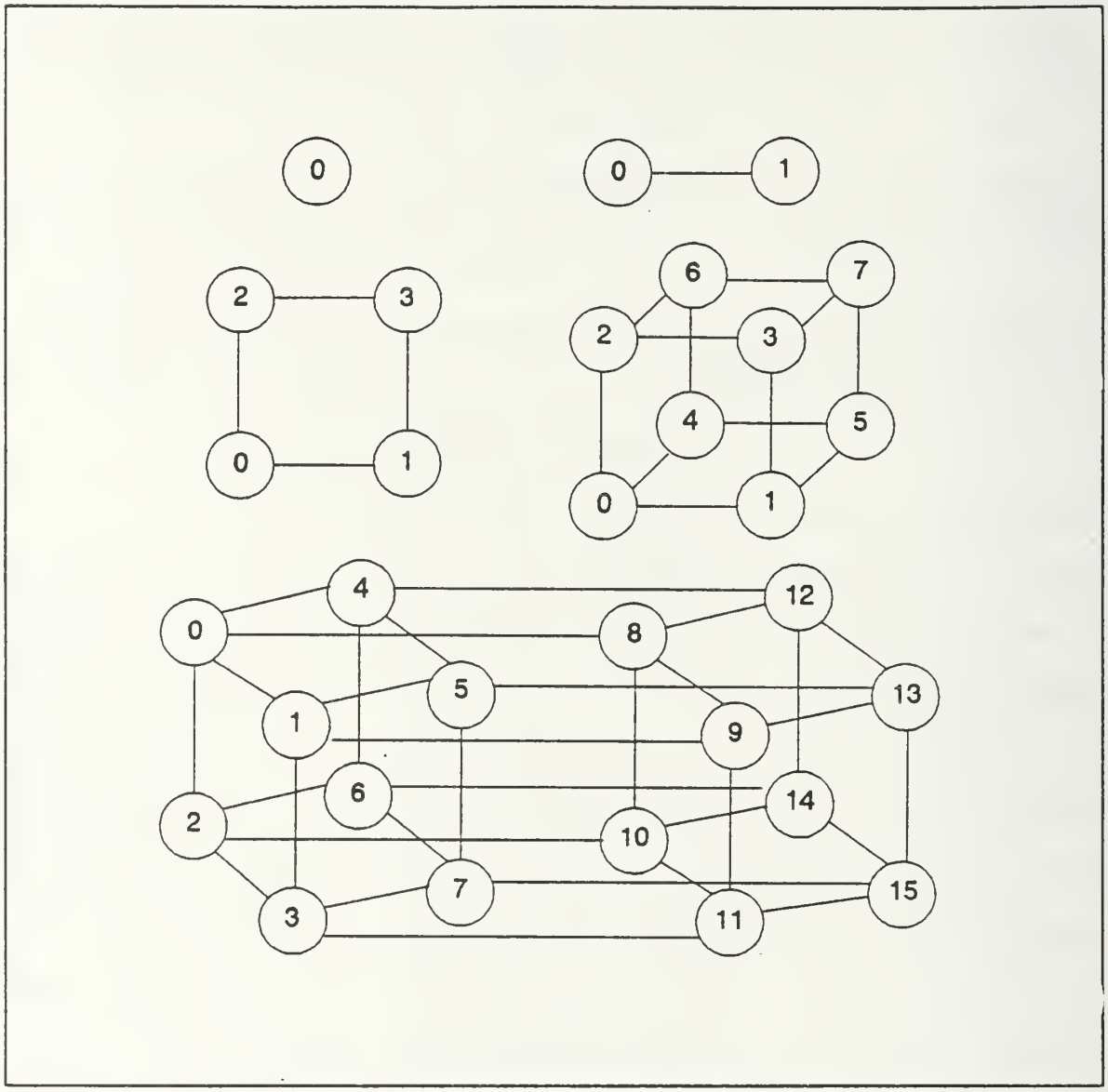

Figure 3.2 Hypercubes of dimension zero through four

time taken to complete the same set of computations exploiting parallelism, T_p,

$$S_p = \frac{T_s}{T_p} \qquad (3.1)$$

Although speedup compares the time taken for the serial computer program and the parallel computer to complete the same set of computations, this set does not imply both programs follow the same algorithm. Parallel programs often contain additional operations to accommodate parallelism. In order not to be misleading, *speedup* should compare the parallel computer program with the most efficient serial computer program. Many suggest that times, T_p and T_*, be measured using a particular parallel computer and the fastest serial computer. However, the variation in the technical specifications of both computers may cloud the issue whether parallel processing is beneficial. To be an effective measure, the computing technical specifications of the individual processor of the parallel computer and the serial computer should be equal. Therefore, for the purpose of this thesis, *speedup*, S_p, is measured by the ratio of time, T_1, taken by the parallel computer executing the most efficient serial algorithm and the time, T_p, taken by the same parallel computer executing the parallel algorithm using p processors.

$$S_p = \frac{T_1}{T_p} \qquad (3.2)$$

The other measure, *efficiency*, accounts for the relative cost of achieving a specific speedup. Relative cost is measured as the number of processors required to achieve the speedup. *Efficiency*, E_p, is the ratio between the

speedup, S_p, and the number of processors, p (the theoretical speedup).

$$E_p = \frac{S_p}{p} \tag{3.3}$$

Many factors could possibly limit the possible speedup and efficiency of a parallel program. These factors include the number of sequential operations that cannot be parallelized, the communication time between individual processors, and the time each processor is idle due to synchronization requirements. Many have argued these factors severely restrict the benefits of parallel computing. Despite these factors, research has shown parallel computing can be an effective means to reduce computation time (see Quinn, 1987, pp. 18 - 20 and Gustafson, 1988). Considering only the number of sequential operations in a program that cannot be parallelized, Amdal's Law states that the maximum speedup, S_p, achievable by p processors is:

$$S_p \leq \frac{1}{f + (1-f)/p} \tag{3.4}$$

where f is the fraction of operations that must be performed sequentially (Amdahl, 1967, pp. 483 - 485). Equation 3.3 provides an initial means to determine if an algorithm is a good candidate for parallelization.

B. INTEL iPSC/2 HYPERCUBE

To maximize speedup and efficiency, parallel algorithms must be developed with a specific parallel computer in mind. In determining the parallel computing potential of the NAVSPASUR satellite motion model, an INTEL iPSC/2 hypercube computer, located at the Department of Mathematics at the Naval Postgraduate School, was used. The iPSC/2 is a MIMD multicomputer with a hypercube topology. The iPSC/2 consists of a system resource manager and eight individual processors, called computing nodes. The system resource manager, often called the host, provides the interface between the user and the computing nodes. The host is a 386-based computer, which may be used to process data in addition to providing the interface for the user.

The computing nodes are complete, self-contained INTEL 80386 microprocessors. Each computing node also contains a 80387 numeric coprocessor, its own local memory, and a Direct-Connect communications module (DCM). Each computing node may be augmented by a Vector Extension (VX) module for pipelined vector operations. The iPSC/2 located at the Naval Postgraduate School contains only one node with the VX module.

Communications among the nodes and the host are completed through message passing. The Direct-Connect Module (DCM) allows messages to be sent directly to the receiving node without disturbing the other node processors. Other hypercube designs require messages to be stored and forwarded along a

path of connected nodes until the message reached the receiving node.

The iPSC/2 uses a UNIX operating system and may be programmed in Fortran and C languages. A more detailed listing of the INTEL iPSC/2 hypercube's technical specifications is contained in Appendix B.

C. METHODS OF PARALLELIZATION

1. Vectorization

Vectorization is one method to parallelize an existing sequential program. *Vectorization* is the process of converting blocks of sequential operations into vector instructions that may be pipelined. A simple example of vectorization using Fortran is the following:

Sequential Code:

```
      Do 10 i=1,N
10    z(i)=x(i)+y(i)
```

Vector Code (VAST2):

```
      call vadd(N,x,1,y,1,z,1)
```

To assist in the vectorization of a serial program, there exist many commercially-available vectorizing compilers. (Quinn, 1987, pp. 233 - 235)

Vectorizing compilers automatically vectorize sequential program code for execution. Additionally, they may

identify to the user program constructs and data dependencies that limit potential vectorization. Vectorization cannot be maximized solely by a compiler. Most vectorizing compilers have a limited ability in recognizing sequential blocks to be vectorized and translations may not be always straight forward. INTEL iPSC/2 contains the vectorizing compiler, VAST2. The VAST2 complier supports only Fortran programs and is limited to vectorizing only *do loops* and *if statements*. (iPSC/2 VAST2 User's Guide, 1989)

2. Distributing Computations

By parallelizing tasks on individual processors, Vectorization provides only the first level of parallelism. In order to partition a program into parallel tasks to distribute among the processors of a multi-computer, a different strategy is needed. Although there exist many commercially-available vectorizing compilers, compilers which identify higher levels of parallelism have not been as successful. Therefore, the task of developing an algorithm to efficiently distribute computations among several processors is left to the user.

Performance of parallel algorithms may be radically different for different parallel computers. A number of factors such as processor speed, memory access time, and memory capacity can affect an algorithm's performance. Hence, the strategy to parallelize an algorithm must be developed

with a specific parallel computer in mind. The multicomputer with each node having its own memory provides the greatest flexibility to the user. For a multicomputer, the user must partition the problem among the processor nodes. The hypercube topology allows the user to use the natural topology of the problem to decompose the problem into parallel *processes*. A *process* is defined as a single statement or a group of statements which are a self-contained portion of the total computations. Using the INTEL iPSC/2, two decomposition strategies are suggested (iPSC/2 User's Guide, 1990. pp. 4-1 - 4-6):

- Control Decomposition
- Domain Decomposition

a. Control Decomposition

Control decomposition is the strategy of dividing tasks or processes among the individual processors (nodes). This strategy incorporates a divide and conquer approach. Control decomposition is recommended for problems with irregular data structures or unpredictable control flows.

One method of control decomposition is for the parallel program to self-schedule tasks. For this method one node assumes the role of a manager with the remaining nodes assuming roles as workers. The managing node maintains a list of processes to be accomplished and assigns a processes to the working nodes. The working nodes request jobs, receive

processes, and perform the indicated tasks. Implied in the self-scheduling method is the cost of one processor to perform the manager duties. (iPSC/2 User's Guide, 1990, p. 4-4)

A second method of control decomposition is to pre-schedule the processes. The exact tasks required of each node are explicitly stated in the parallel program. Although this method saves the cost of the managing node, care must be taken to ensure the processes are evenly distributed among the nodes.

b. Domain Decomposition

Domain decomposition is the strategy of dividing the input data or domain among the nodes. The partitioned sets of domain may be specific data sets such as blocks of matrix or represent a specific grid such as used in finite difference or finite element methods to solve partial differential equations. The major difference between control and domain composition is that domain decomposition strategy requires each node to perform essentially the same tasks but with different input data.

Domain decomposition is recommended if the calculations are based on a large data structure and the amount of work is the same for each node. An example of domain decomposition is multiplying two large matrices by block multiplication. Although domain decomposition may seem perfectly parallelizable and thereby very efficient, user must

use caution to ensure each input data set requires essentially the same amount of work.

3. Improving Performance

The decomposition of a problem may require the use of the control, domain or a hybrid of both strategies to be efficient. Once a specific strategy is chosen, several factors should be considered to improve the performance of the parallel algorithm. Those factors include:

- Load balance
- Communication to computation ratio
- Sequential bottlenecks

a. Load Balance

Load balance refers to the degree to which all nodes are active. If the work is not evenly distributed among the nodes, the parallel algorithm will show constrained speedup. Load balancing may be achieved by decreasing the grain size of the parallel tasks, self-scheduling tasks, or redistributing the domain. *Grain size* refers to the relative amount of work completed in parallel. Pipelined vector operations is an example of small grain parallel computing and distributing computations may be considered as large grain parallel computing.

b. Communication to computation ratio

Communications to computation ratio is the ratio between the time spent communicating and the time spent

computing. Except for perfectly parallel problems, time lost for communications is inherent in parallel algorithms. A large communication to computation ratio constrains a parallel program's performance. The objective is to maximize the time a node spends computing and to minimize the time spent communicating. Reductions in the communication to computation ratio may be accomplished by increasing the grain size, grouping messages, or recalculating values instead of receiving the value from another node.

c. Sequential Bottlenecks

Sometimes tasks cannot begin until completion of a previous task, limiting number of tasks that can be completed in parallel. A *sequential bottleneck* is the circumstance of other processors waiting for another processor to complete a task before they may continue. The portion of operations that are not completed in parallel can substantially restrict speedup as can be seen by Amdahl's Law (Equation 3.3). Inherent in sequential bottlenecks are any requirements of the nodes to synchronize. The only method to remove sequential bottlenecks is to modify or reorder the algorithm in order to overlap sequential code with other computations.

IV. PARALLELIZATION OF PPT2

The purpose of this research was to determine the potential reduction in computation time for the NAVSPASUR satellite motion model through parallel computing. This potential may be assessed by determining the relative speedup and efficiency of various parallel algorithms employing the methods and strategies of parallelization discussed in Chapter III.

As stated in the previous chapter, the strategy for developing parallel algorithms depends heavily on the architecture and topology of the parallel computer used. Due to ease of access and familiarity with the INTEL iPSC/2 hypercube, the parallel computing potential of the NAVSPASUR model was assessed with respect to implementing the model on this specific multi-computer. Although performance of various algorithms may vary somewhat depending on the specific parallel computer, it was hoped that some generalizations may be made from the application of the NAVSPASUR model to this hypercube.

A. VECTORIZATION

The first method of parallelization considered for the NAVSPASUR model was vectorization. Vectorization is usually simpler than the other methods of parallelization to apply.

Additionally, if vectorization proved to be beneficial, it may be incorporated with the other parallel computing methods in order to realize even greater speedup and efficiency.

The realized speedup due to vectorization is a function of the number of vector operations within a specific algorithm. Vector operations in Fortran are usually characterized by *do loops* containing scalar operations performed on each element of an array. With each node possessing its own vector co-processor (VX module), these *do loops* may be replaced by single calls to canned subroutines. These subroutines utilize a vector co-processor to perform pipelined vector operations, significantly reducing computation time. In addition to the explicit vector operations within an algorithm, sometimes there exist blocks of scalar operations that may easily be transformed into vector operations. Scalar operations contained within Fortran *do loops* and logical *if statements* are usually good candidates.

Analysis of the Fortran subroutine **PPT2** shows that the current subroutine contains very few explicit or implicit vector operations. The only apparent vector operation in the satellite state vector prediction portion of **PPT2** is the computation of the velocity vector at the very end of the algorithm. The propagation of the orbital element set comprises the majority of the computations. The formulas used to propagate the orbital elements, presented in Chapter II, may be characterized as lengthy, algebraically-complex, non-

linear scalar functions of the mean orbital elements. Attempts to transform these formulas into a set of vector operations quickly become algebraically overwhelming and a successful transformation is highly improbable.

Likewise, with the exception of the computation of the variable T2, the scalar operations contained in the *do loops* and *if statements* of **PPT2** demonstrate limited vectorizing potential due to data dependency within the loops and statements. Therefore, based on this initial assessment of limited vectorizing potential, vectorization was not considered as a viable method to reduce computation time and efforts to vectorize **PPT2** were pursued no further.

B. DISTRIBUTION OF COMPUTATIONS

With vectorization deemed as not a viable method of parallel computing for the NAVSPASUR model, any reduction in computation time needed to be achieved through the method of distributing computations. Both strategies of *control decomposition* and *domain decomposition* were considered.

In order to better appraise the potential reduction of computation time by implementing each strategy, separate parallel algorithms utilizing the different strategies were developed and evaluated with respect to the measures of speedup and efficiency. Although a combination of both strategies may possibly provide the greatest speedup and efficiency, the evaluation of separate algorithms implementing

the respective strategies exclusively provided a better means to determine how the majority of the reduction in computation time was achieved. Additionally, once the relative benefit of each strategy is determined, it would not be difficult to incorporate both algorithms together on the hypercube.

Using the two distinct strategies, two separate sets of programs were developed and evaluated. Each program set consists of two Fortran programs to be executed on the INTEL iPSC/2. A host (system resource manager) program acquires a specified size cube; loads the node program on the processors of the attached cube; and, upon completion of the algorithm, releases the cube for another application. The node program implements the parallel algorithm. Although the host may also serve as an additional processor, the parallel algorithms utilized only the nodes of the iPSC/2.[5]

Program set named **P^{3}T-4** implements an algorithm using the control decomposition strategy and the program set named **P^{3}T** implements an algorithm using the domain decomposition strategy. Descriptions of the algorithms and an assessment of their respective results are contained in the subsections below.

---

[5]The routine used to determined run times for the various programs is measured differently for the host and the nodes. In order to obtain comparable times to compute speedup and efficiency, the actual parallel algorithms utilize only the node processors. (iPSC/2 Programmer's Reference Manual, 1990, p. 3-174.)

1. Control Decomposition -- P^3T-4

The strategy of control decomposition is to reduce the NAVSPASUR model's computation time by the concurrent completion of separate tasks (processes) by the individual nodes of the hypercube. By reducing the computation time necessary to predict each individual satellite's state vector, an overall reduction in computation time to predict state vectors for all tracked objects can be achieved. Hence, the ultimate objective of the program set, **P^3T-4**, was to reduce the computation time for a single object in orbit.

a. Algorithm

In order to predict a satellite's state vector considering the secular and periodic correction terms due to the zonal harmonics and a correction term for each element due to the sectoral harmonics, the NAVSPASUR model requires the completion of 55 major tasks. The majority of tasks are evaluation of the formulas outlined in Chapter II, some tasks are a group of computations such as the group of computations necessary to compute the variable T2 or the group of computations to solve Kepler's Equation by Steffensen's Method.

The first step in partitioning these tasks among the nodes was to determine which tasks could be completed concurrently. Concurrency was determined by the development of a hierarchy of the formulas used by the NAVSPASUR model.

Each of the individual tasks were listed with its respective required input. Tasks which required output from the completion of other tasks were listed below those tasks. Tasks which could be executed concurrently were listed on the same level of the hierarchy. Figures 4.1 and 4.2 contain an extract of this hierarchy.

From this hierarchy of formulas, the number of tasks that could be completed concurrently at each level of the hierarchy ranges from 2 to 14. The levels where only a few (less than four) represent potential sequential bottlenecks. These sequential bottlenecks needed to be overcome for a high level of efficiency to be achieved.

Additionally, the number of FLOPS required varied considerably among the tasks. Some tasks required as few as 2 FLOPS, while other tasks required over 200 FLOPS. For example, solving Kepler's Equation by Steffensen's Method could require as few as 3 FLOPS or as many as 650 FLOPS based on the speed of convergence.[6] This variance in the number of FLOPS required by the various tasks presented a potential problem in load balancing.

The second step in applying this strategy was to determine the method of scheduling the tasks to be accomplished among the nodes. A manager-worker algorithm as described in Chapter III provides an easy method to achieve

---

[6]If the error tolerance of less than 10^{-8} is not met, the NAVSPASUR model halts Steffensen's Method after 20 iterations.

Level	Variable (Equation)						
1	T2 (2.47-2.48)	a_0" (2.50)					
2	γ_1' (2.54)	á (2.41)	é (2.45)	Δl (2.38)			
3	dg/dl (2.51)	dh/dl (2.52)	l" (2.55)	a" (2.57)	e" (2.57)	VLE1 (2.59)	VLE2 (2.59)
4	g" (2.55)	h" (2.55)	cos f" (2.58)	VLL2 (2.59)			
5	δ_1e (2.59)	eδ_1l (2.59)	r" (2.58)	sin f" (2.58)	sinI" δ_1h (2.59)	δ_1z (2.67)	δ_2e (2.61)
6	δ_1I (2.59)	eδl (2.64)	sin(½I) δh (2.66)	δ_2z (2.67)	δe (2.62)	a (2.69)	
7	δI (2.64)	e (2.70)	l (2.70)	z (2.68)			
8	cos I (2.70)	h (2.70)	cos f (2.72)				
9	g (2.71)	r (2.72)					
10	$\dot{r}$ (2.37)	$\dot{v}$ (2.37)					
11	$\bar{r}$ (2.37)	$\bar{v}$ (2.37)					

Figure 4.1 Hierarchy of NAVSPASUR Formulas

load balancing among the nodes. However, for a large number of small tasks as is the case with the NAVSPASUR model, this type of algorithm can become communication intensive. A large communication-to-computation ratio can severely limit speedup and could possibly cause a parallel algorithm run longer than the original serial algorithm. In an effort to reduce the

Level	Variable (Equation)						
1							
2							
3	VLE3 (2.59)	VLH1I (2.59)	VLH2I (2.59)	VLH3I (2.59)	VLS1 (2.60)	VLS2 (2.60)	VLS3 (2.60)
4							
5	$e\delta_2 l$ (2.63)	sinI" $\delta_2 h$ (2.65)	$\delta_2 a$ (2.18)	sectoral terms (LUNAR)			
6							
7							
8							
9							
10							
11							

Figure 4.2 Hierarchy of NAVSPASUR Formulas (continued)

amount of communication in the algorithm, an algorithm that pre-scheduled tasks was chosen.

With pre-scheduled algorithms, each node knows its own tasks to accomplish without communicating with a "manager" node. Additionally, the absence of a "manager" node frees one more node to assist in completing the required tasks. Despite

these positive points, pre-scheduled algorithms are not without their own drawbacks. Pre-scheduling algorithms do not provide automatic load balancing as is the case with self-scheduling algorithms. Care must be taken to ensure each node does essentially the same amount of work. Also, pre-scheduling algorithms require a fixed number of nodes. Requiring a fixed number of nodes restricts the flexibility of algorithm and its potential speedup.

The third step in applying this strategy was to determine the number of nodes for pre-scheduling. Factors in determining the number of nodes or cube size include the potential requisite speedup, the amount of computation completed between communication messages, potential sequential bottlenecks, the number of messages required, and efficient use of all of the nodes. In an attempt to achieve an appreciable speedup, the algorithm was developed to use a minimum of four nodes.

The final step in applying this strategy was assigning specific tasks to each node. Load balancing and potential synchronization problems were considered in the assignment of the tasks to the respective nodes. Often, communication distances are also considered in assigning tasks to the nodes; however, with such a small number of nodes the communication distances were negligible. The diagram in Figure 4.3 depicts how the tasks were distributed among the four nodes. Some of the smaller initial tasks were duplicated

by several nodes in order to limit the amount of communication and eliminate potential synchronization problems at the beginning of the algorithm. A complete listing of the source code for program set, **P^{3}T-4**, is contained in Appendix C.

b. Assessment

To establish a baseline to compare the performance of the parallel program sets with the serial subroutine **PPT2**, execution times for **PPT2** to predict the position of a single satellite and a set of satellites, ranging from 12 to 20736, were measured. The measured times were the elapsed time for the execution of **PPT2** in milliseconds on a single node using the node's internal clock. Using ten different sets of satellite data, the mean execution time for propagating a single satellite was 11.2 milliseconds. The mean execution time for **PPT2** to propagate 12 to 20736 satellites is depicted in Figure 4.4. These mean execution times were used to compute the speedup and efficiency of both parallel program sets.

(1) Results. Program set **P^{3}T-4** was executed with the same sample satellite sets as were used with **PPT2**. The graph in Figure 4.5 shows a comparison of the mean executions of **P^{3}T-4** and **PPT2** for a various number of satellites. As one can see, **P^{3}T-4** was nearly two times slower than the original serial subroutine.

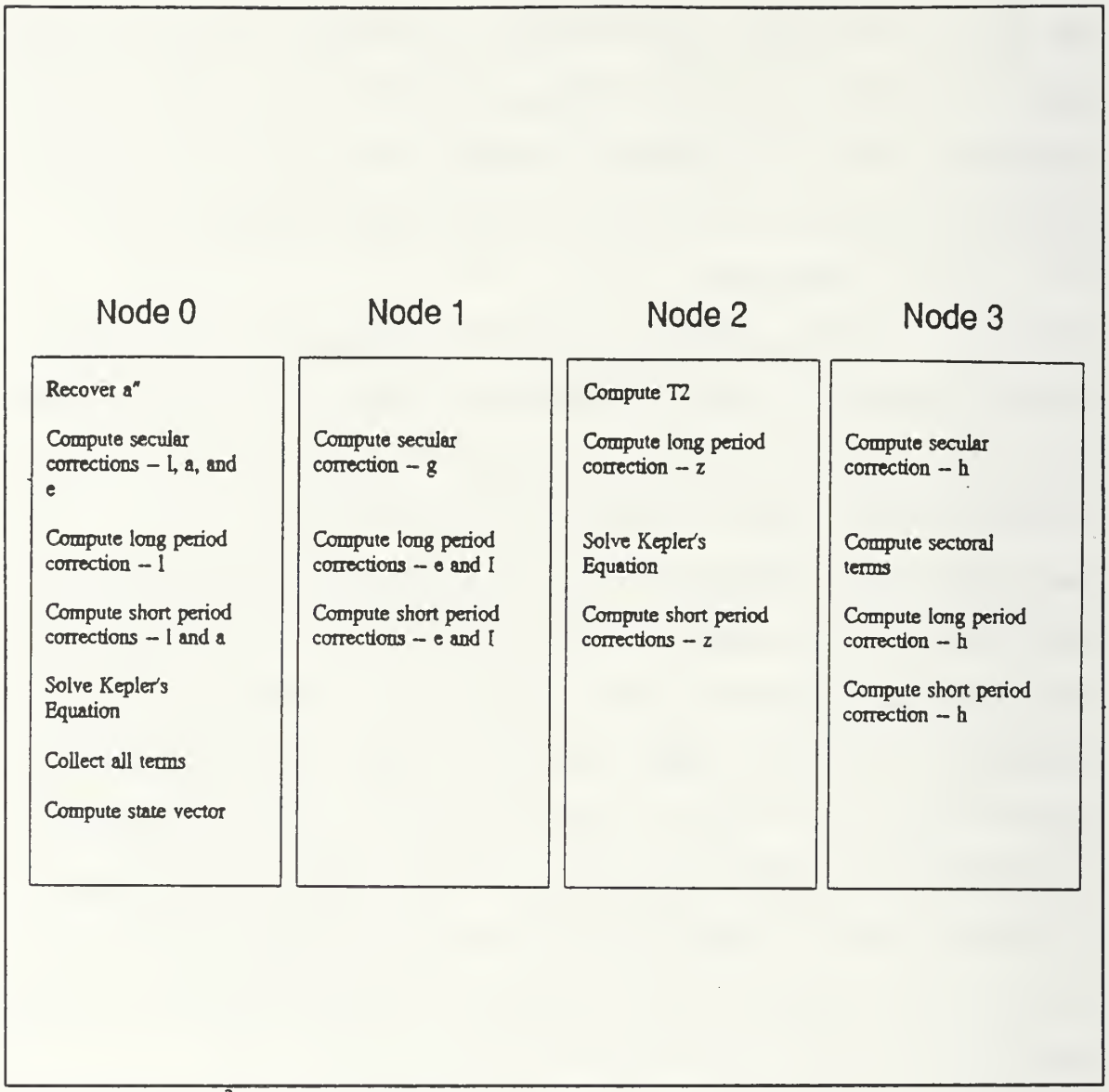

Figure 4.3 P^{3}T-4 Algorithm

For a single satellite, the mean execution time for **P^{3}T-4** was 23.3 milliseconds as compared to only 11.2 milliseconds for **PPT2**. A closer look at where the time is spent reveals the shortcomings of this parallel algorithm. Table 4.1 shows a comparison of mean execution times for the one node executing **PPT2** and the four nodes executing **P^{3}T-4**.

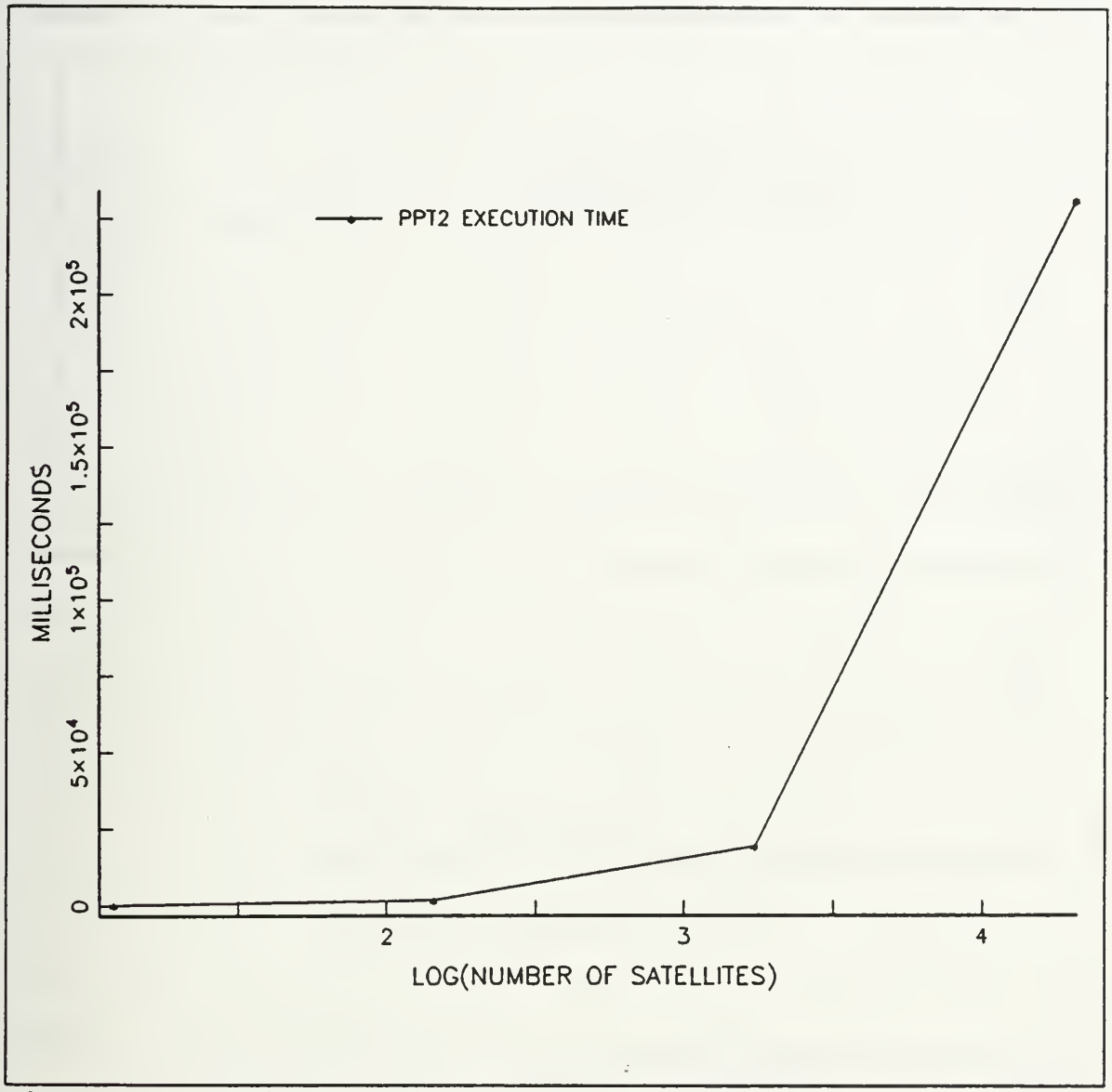

Figure 4.4 PPT2 Execution Times

The times expressed in the table are the mean for the ten sample satellites. The communication time, t_m, includes the time spent sending and receiving messages, plus the time spent waiting for messages to arrive. The computing time, t_c, is the time each node spent completing its respective tasks.

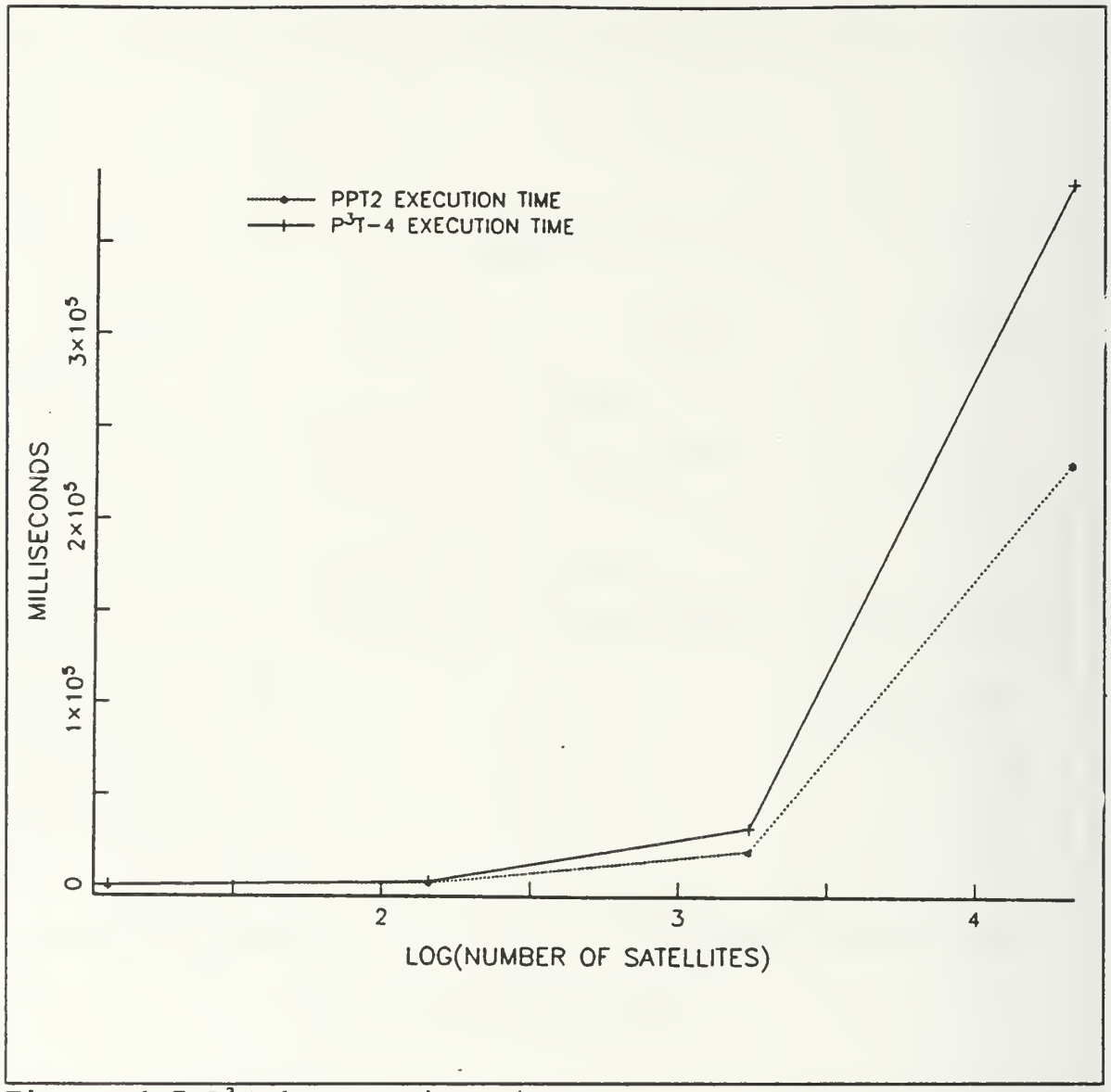

Figure 4.5 P^{3}T-4 Execution Times

As seen by the times listed in Table 4.1, two problems with the algorithm become evident. First, communication time outweighs the actual computation time for each node. The causes of the long communication time are number of messages required by this specific partition of tasks and synchronization problem of nodes waiting to receive

Table 4.1 P³T-4 Execution Time Breakdown

Algorithm	t_c (milliseconds)	t_m (milliseconds)	t (milliseconds)
PPT2			
(one node)	11.2	NA	11.2
P³T-4			
node 0	4.3	19.0	23.3
node 1	2.2	15.9	18.1
node 2	2.7	14.7	17.4
node 3	5.8	15.7	21.5

computed values from other nodes. Second and most importantly, although the actual computation time was reduced, the total execution time of the parallel algorithm is longer than the serial algorithm implemented by **PPT2**.

(2) Improvements. The major source of the problem is the communication to computation ratio. This parallel algorithm using four nodes requires 23 messages among the nodes. The NAVSPASUR model is not computationally intensive enough to offset this amount of communication. To improve performance this ratio must be reduced.

One method to reduce the communication to computation ratio is to reduce the amount of communication. One way to reduce the number of communications is to restructure the partitions. However, other partitions using four nodes were analyzed, yet none could significantly reduce

the number of messages. One alternative to possibly achieve any speedup was to partition the computations among fewer nodes. The diagram in Figure 4.6 depicts the distribution of tasks for a two node algorithm **P³T-2**. Although the algorithm displayed potential in reducing the total execution time to less than **PPT2**, speedup would be further bounded by two. A speedup of two would not outweigh the costs in procuring a parallel computer merely for satellite propagation.

A second alternative for reducing the communication to computation ratio is to somehow increase the amount of computation between messages. The amount of computation could be increased by computing the intermediate values for *n* satellites in an array and sending the array in one message. The communication would remain essentially constant and the computation between messages would increase by a factor of *n*. An estimate of this improvement may be made for the mean times in Table 4.1 using speedup and efficiency. From Chapter III, efficiency is expressed as the following:

$$E = \frac{S_p}{p} = \frac{t(1)}{pt(p)} \qquad (4.1)$$

where

$$t(p) = t_c(p) + t_m(p) \qquad (4.2)$$

If E_n is the efficiency of the **P³T-4** algorithm computing *n* satellites' values between messages, then

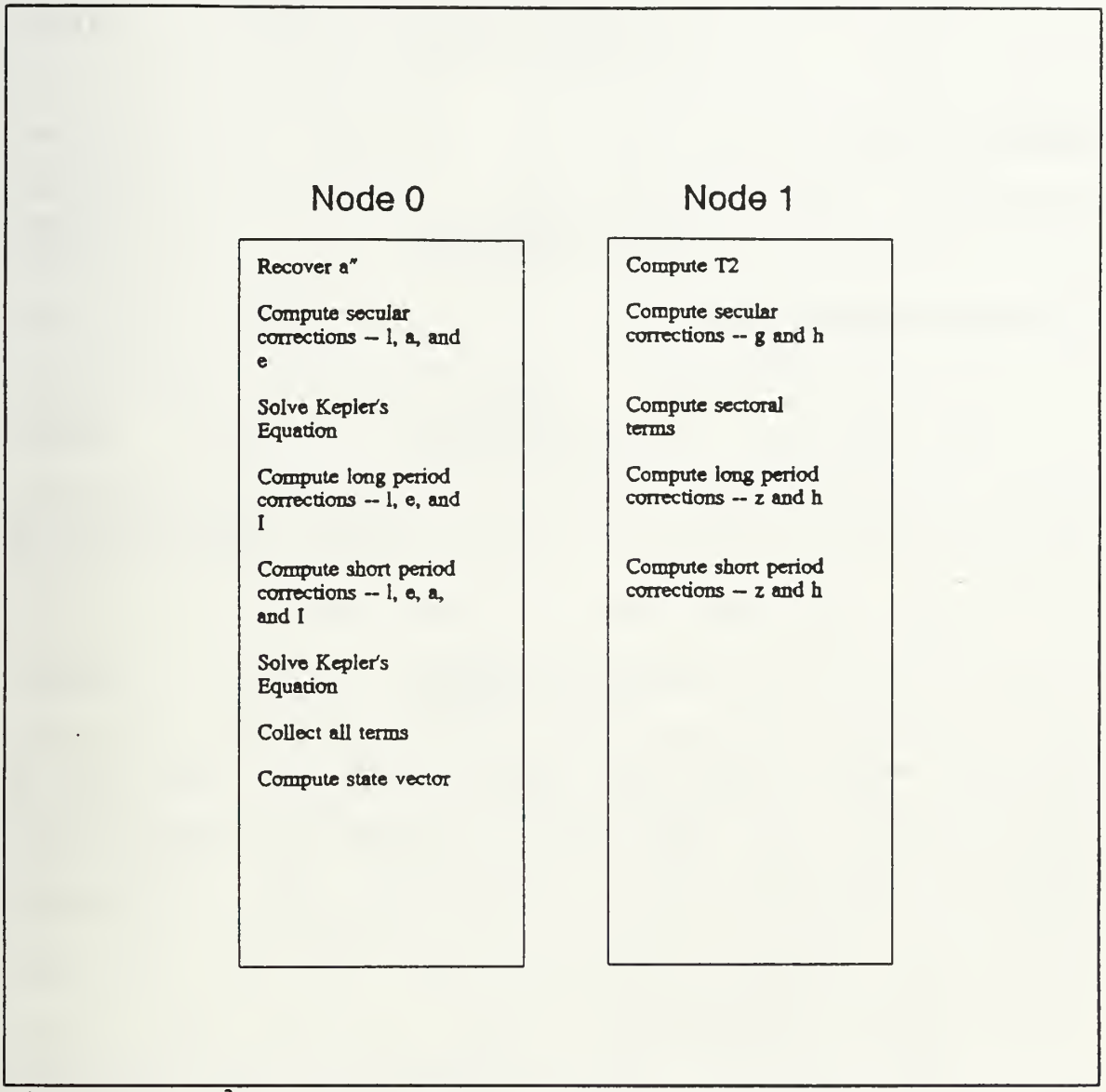

Figure 4.6 P³T-2 Algorithm

$$E_n = \frac{nt(1)}{p(nt_o(p) + t_m(p))} \qquad (4.3)$$

Solving Equation 4.1 for t(1) and substituting into Equation 4.3 yields

$$E_n = \frac{n(t_c(p) + t_m(p))}{(nt_c(p) + t_m(p))} E \qquad (4.4)$$

Replacing E by

$$E = \frac{t(1)}{p(t_c(p) + t_m(p))} \qquad (4.5)$$

and simplifying yields

$$E_n = \frac{t(1)}{p(t_c(p) + \frac{t_m(p)}{n})} \qquad (4.6)$$

Take the limit as n goes to infinity and the upper bound for E_n is

$$\lim_{n \to \infty} E_n = \frac{t(1)}{pt_c(p)} \qquad (4.7)$$

Setting p equal to four and using values from Figure 4.1, E_n is bounded by .48. This implies the maximum speedup of the modified algorithm would be bounded by 1.92. Again, the benefit of using this strategy is quite limited.

2. **Domain Decomposition -- P^{3}T**

The strategy of domain decomposition is to reduce the NAVSPASUR model's computation time by the concurrent computation of several satellites' state vectors. Each node of the hypercube would complete identical tasks on different satellite data sets, simultaneously. Hence, the ultimate objective of program set **P^{3}T** was to reduce the overall computation time for several objects in orbit.

a. Algorithm

Unlike the application of the control decomposition strategy, the application of the domain decomposition strategy to the NAVSPASUR model was seemingly less arduous. First, because each node propagates satellite data sets independent of the other nodes, there exists no requirement for communication or synchronization among the nodes. This lack of communication simplifies the load balancing and sequential bottleneck problems present in the **P^{3}T-4** parallel algorithm.

Second, because each node may perform the satellite state vector prediction tasks serially, the existing subroutine **PPT2** may be used with only minor modifications. Developing a parallel algorithm for predicting an individual satellite's state vector was a major task for the control decomposition strategy. Additionally, by using the existing **PPT2** code, the other tasks completed by **PPT2** may be requested by the user using the same control variables as used by the original **PPT2** subroutine. The **P^{3}T-4** program set was restricted to only predicting a satellite's state vector.

Finally, by using the serial subroutine **PPT2**, this strategy may be reduced to only developing an algorithm to distribute the data in a timely manner. Maximum efficiency will be achieved if the nodes do not have to wait for satellite data to propagate.

Intuitively, this strategy seems perfectly parallelizable. Although the various tasks performed by **PPT2** require different computation times, the total execution time for each node will be essentially the same if it is assumed that the various tasks are randomly distributed throughout the input data sets. The concern for this algorithm was the potential sequential bottlenecks at input/output portions of the program set. Reading and writing to external files can be very time consuming. In addition to the actual time spent reading/writing to an external file, a certain amount of time is spent to access the file. In order to minimize this time, the number of calls to read/write to a file should be minimized.

With the specific iPSC/2 hypercube available, input/output is completed sequentially. Each node must compete with the other nodes to read and write to external files. To minimize time lost to accessing the file cataloging the set of satellites, a node was devoted to both the reading/distributing of input satellite data and to the collecting/writing of the results. The idea of using a single node to read the data and a single node to subsequent write the output is simple to implement and proved to be fastest method to overcome the bottlenecks with the input/output. The remaining nodes of the hypercube implement the NAVSPASUR model using a slightly modified **PPT2**. The diagram in Figure 4.5 depicts how the satellite data is distributed. The cost of

using this simple algorithm to distribute and collect the data is the loss of two nodes. The only restriction on the size of the hypercube required by **P^{3}T** is that the attached cube must contain at least four nodes to achieve any speedup. A complete listing of the source code for program set, **P^{3}T**, is contained in Appendix D.

b. Assessment

(1) Results. The graph in Figure 4.8 depicts the mean execution time for **P^{3}T** versus the number of satellites propagated using hypercubes of four and eight nodes respectively. **P^{3}T** was successful in reducing the overall execution time to propagate several satellites. Table 4.2 shows the speedup and efficiency of **P^{3}T** for a various number of satellites. As seen in Table 4.2, the speedup achieved using all eight nodes of the hypercube was approximately three times larger than the speedup achieved using four nodes. With this parallel algorithm using six "working" nodes for an eight processor hypercube and only two "working" nodes for a four processor hypercube, an increase in speedup by approximately a factor of three was expected. More notable was the increase in efficiency using eight versus four nodes. The efficiency increased from .45 to .67. This increase in efficiency indicates that **P^{3}T** applied to a hypercube of greater dimension could yield even greater speedup and efficiency.

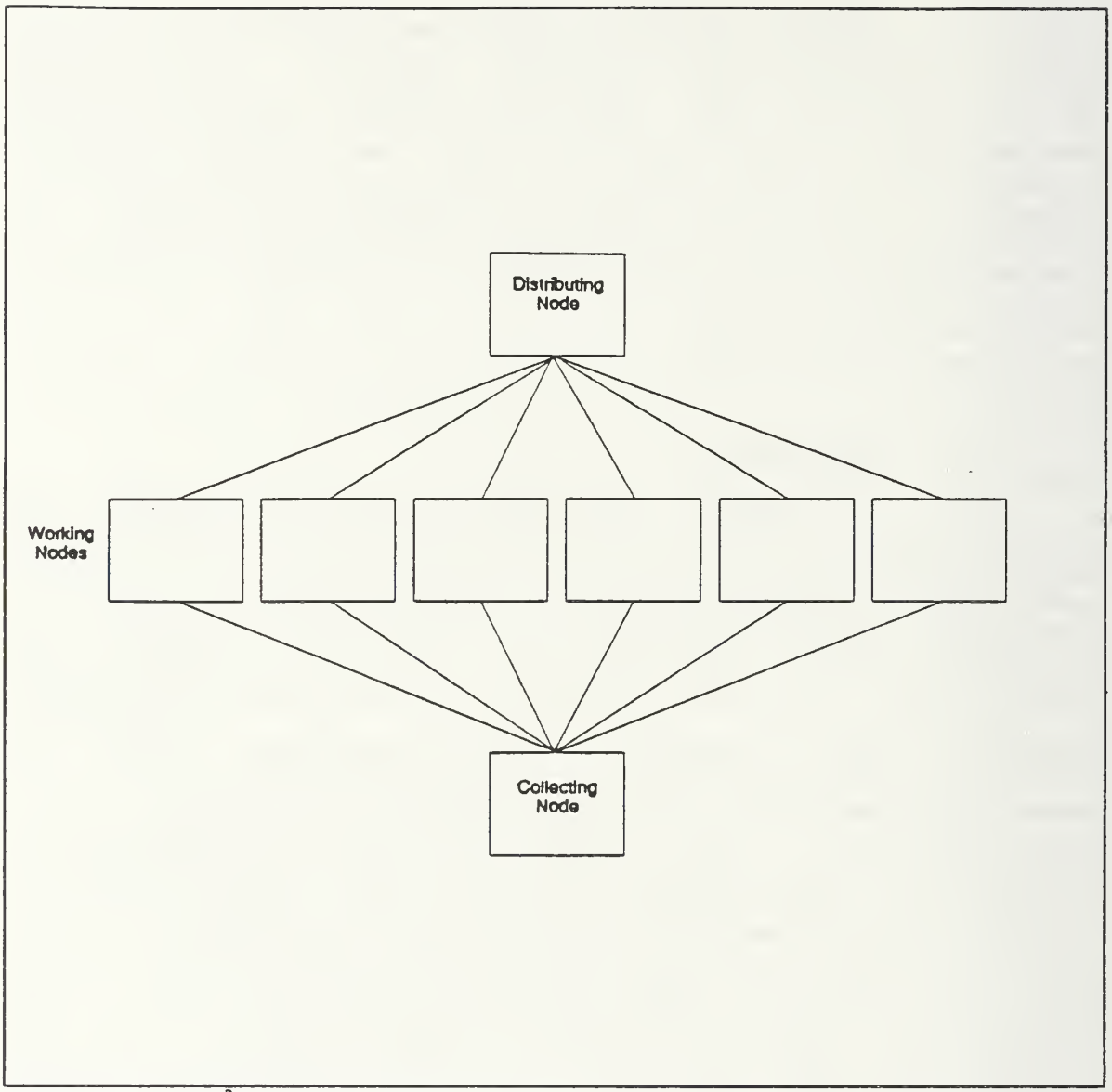

Figure 4.7 P^3T Algorithm

Table 4.2 also indicates that P^3T performance increased somewhat with an increase in the total number of satellites propagated. Because with this parallel algorithm the computation to communication ratio does not vary with the number of satellites, this small increase in performance must be primarily due to the diminishing impact of the algorithm's

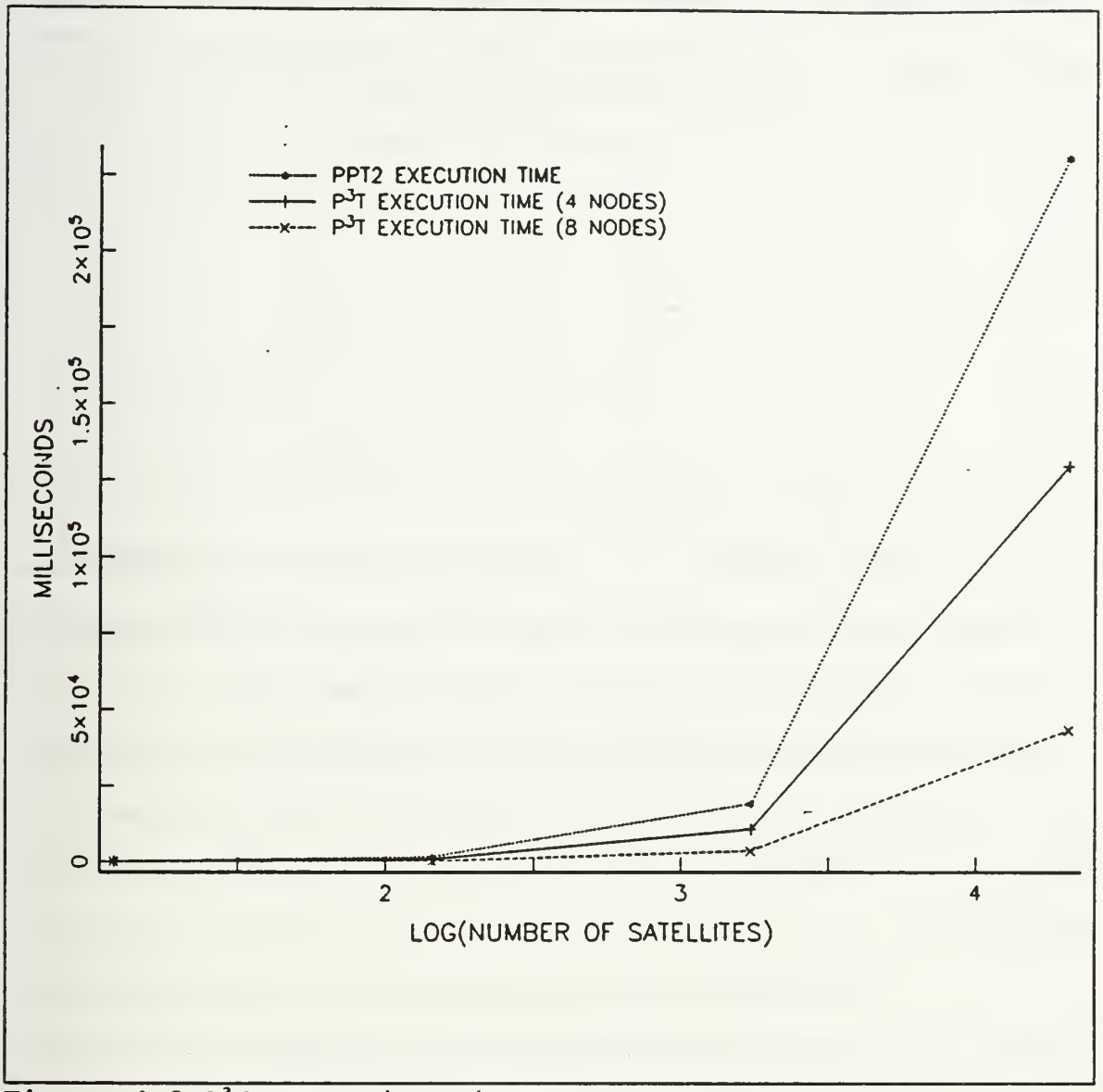

Figure 4.8 P^3T Execution Times

overhead on total execution time. This overhead includes one additional message containing the total number of satellites to propagate from the distributing node to the other nodes; some small computations by working nodes to determine number of data sets to receive; and a halting message sent by the collecting node to the host once all of the nodes are

Table 4.2 P³T Performance (satellite position prediction)

P³T	Number of Satellites	S_p	E_p
8 nodes	20736	5.31	.66
	1728	5.37	.67
	144	5.24	.66
	12	4.13	.52
4 nodes	20736	1.78	.45
	1728	1.80	.45
	144	1.79	.45
	12	1.66	.33

finished. Because these additional messages and computations are only completed once in the program, the time cost associated with this overhead becomes negligible as the number of satellites propagated is increased. The speedup and efficiency remained fairly constant for greater than 144 satellites.

To estimate the impact of increasing the amount of computation on **P³T**'s speedup and efficiency, the execution time to predict a satellite's position and compute the partial derivatives of the orbital elements was also measured for **PPT2** and **P³T**. These results are summarized in Table 4.3. Both speedup and efficiency improved with this increase in computation.

Table 4.3 P^3T Performance (satellite position prediction plus computation of partial derivatives)

P^3T	Number of Satellites	S_p	E_p
8 nodes	20736	5.48	.69
	1728	5.53	.69
	144	5.45	.68
	12	4.82	.60
4 nodes	20736	1.84	.60
	1728	1.86	.47
	144	1.86	.47
	12	1.82	.46

(2) Improvements. The performance results of this algorithm using only four and eight nodes indicated a potential increase in both speedup and efficiency if this algorithm could be applied to a hypercube of greater dimension. Because the number of working nodes is not fixed for this algorithm, P^3T could be applied easily to any size hypercube with no modifications. The efficiency of the algorithm should increase with the cube dimension until the time to distribute a separate satellite data to each working node exceeds the time required by node to propagate a single satellite. Therefore, a possible improvement in the algorithm's performance can be achieved by applying the algorithm to an optimal dimension hypercube.

Because the hypercube at the Naval Postgraduate School is limited to eight nodes, a model was used to estimate the optimal hypercube dimension. The total execution time for **P³T** to propagate n satellites with p processors, t(p), can be modeled by the following expression:

$$t(p) = t_{w1}(p) + t_{w2}(p) + t_c(p) \qquad (4.8)$$

where $t_{w1}(p)$ is the time the last node must wait to receive its first satellite data set, $t_{w2}(p)$ is the total time the last node must wait to receive all of its subsequent satellite data sets, and $t_c(p)$ is the time for each node to propagate each share of the n satellites.

As described in previous chapter, the iPSC/2 uses a Direct-Connect Module (DCM). This module provides an essentially constant startup time for a message to be passed between two nodes regardless of the length of the message path. Hence, the time to send a message between two nodes is a function of only the size (number of bytes) of the message. Because all messages between the distributing node and working nodes are of a constant size (674 bytes), the time of a single message between the distributing node and each working node is essentially constant. For this algorithm, there are p-2 working nodes. Denoting the time to send a single message between the distributing node and a working node as $t_m(1)$, the $t_{w1}(p)$ may be modeled by the following:

$$t_{w1}(p) = [(p-2)-1]\,t_m(1) = (p-3)\,t_m(1) \qquad (4.9)$$

In order to determine $t_m(1)$, several experiments were run using the specific iPSC/2 located at the Naval Postgraduate School. The mean value of $t_m(1)$ was approximately .693 milliseconds.

A working node's total wait time for subsequent satellite data sets, $t_{w2}(p)$, is a function of the elapsed time for the working node to propagate a single satellite, the elapsed time for the distributing node to send a subsequent satellite data set to the working node, and the number of satellites the working node must propagate. Because the distributing node distributes the data while the working nodes are computing, the wait time is zero if the subsequent satellite data arrives before the working node is ready to receive.[7] If the subsequent satellite data arrives after the node is finished with the previous satellite, the wait time is the difference between the computing time for the previous satellite and the elapsed time for the distributing node to send the node another satellite data set. Because the distributing node must send a data set to each of the other working nodes prior to sending a subsequent data set to the last node, the elapsed time for the distributing node to send another data set may be also modeled by $t_{w1}(p)$ in Equation 4.9.

---

[7]If a node is not ready to receive a message from another node, the message is stored in a local buffer. The time to read from this buffer is negligible.

The total wait time is then the wait time for each subsequent satellite data set multiplied by the number of data sets received by each working node. Hence, by letting t1 represent the time to propagate a single satellite, $t_{w2}(p)$ may be modeled by the expression:

$$t_{w2}(p) = \begin{cases} 0 & if\ t_{w1}(p) < t1 \\ \left(\frac{n}{p-2} - 1\right)(t_{w1}(p) - t1) & if\ t_{w1}(p) \geq t1 \end{cases} \qquad \textbf{(4.10)}$$

Assuming the time for one node to propagate n satellites, t(1), is

$$t(1) = n(t1)$$

The total computation time for each working node, $t_c(p)$, may be approximated by the continuous function:

$$t_c(p) = \frac{n(t1)}{p-2} \qquad \textbf{(4.11)}$$

Substituting Equation 4.8 into Equations 3.2 and 3.3, the speedup and efficiency using a total of p processors may be modeled by the following expressions:

$$S_p = \frac{t(1)}{t(p)} = \frac{n(t1)}{t_{w1}(p) + t_{w2}(p) + t_c(p)}$$
$$E_p = \frac{S_p}{p} = \frac{n(t1)}{p[t_{w1}(p) + t_{w2}(p) + t_c(p)]} \qquad \textbf{(4.12)}$$

Setting t1 equal to 11.2 milliseconds and $t_m(p)$ equal to .693 milliseconds, t(p), S_p, and E_p were computed using Equations

4.9 - 4.12. Figures 4.9 and 4.10 depict the estimates of $t(p)$, S_p, and E_p for propagating 1728 satellites using 4 to 1024 processors (a cube dimension of 2 to 10). Using the above model, **P^{3}T** is capable of achieving a maximum speedup over 16 with a corresponding efficiency of approximately 90 percent for a hypercube of dimension 5 (2^5 nodes). A hypercube of dimension 4 achieves a speedup of nearly 14 and an efficiency of almost 90 percent. Although these graphs are only estimates of the actual values of speedup and efficiency, they correspond closely to the actual timed results for four and eight node size hypercubes and provide a good indication of the parallel computing potential of this algorithm for higher dimension hypercubes.

Another possible improvement to this domain decomposition algorithm is to eliminate the need for the distributing and collecting nodes. Although the iPSC/2 located at the Department of Mathematics, Naval Postgraduate School is not capable of concurrent input and output, concurrent file systems are available. Separate I/O nodes allow the computing nodes of a hypercube to concurrently read and write to external files. A concurrent file system would eliminate the need of the distributing and collecting nodes. Additionally, the INTEL Concurrent File System (CFS) allows for a common file pointer to be maintained among the computing nodes, minimizing overhead in algorithm. Depending on the efficiency of the I/O nodes, a further increase in the overall

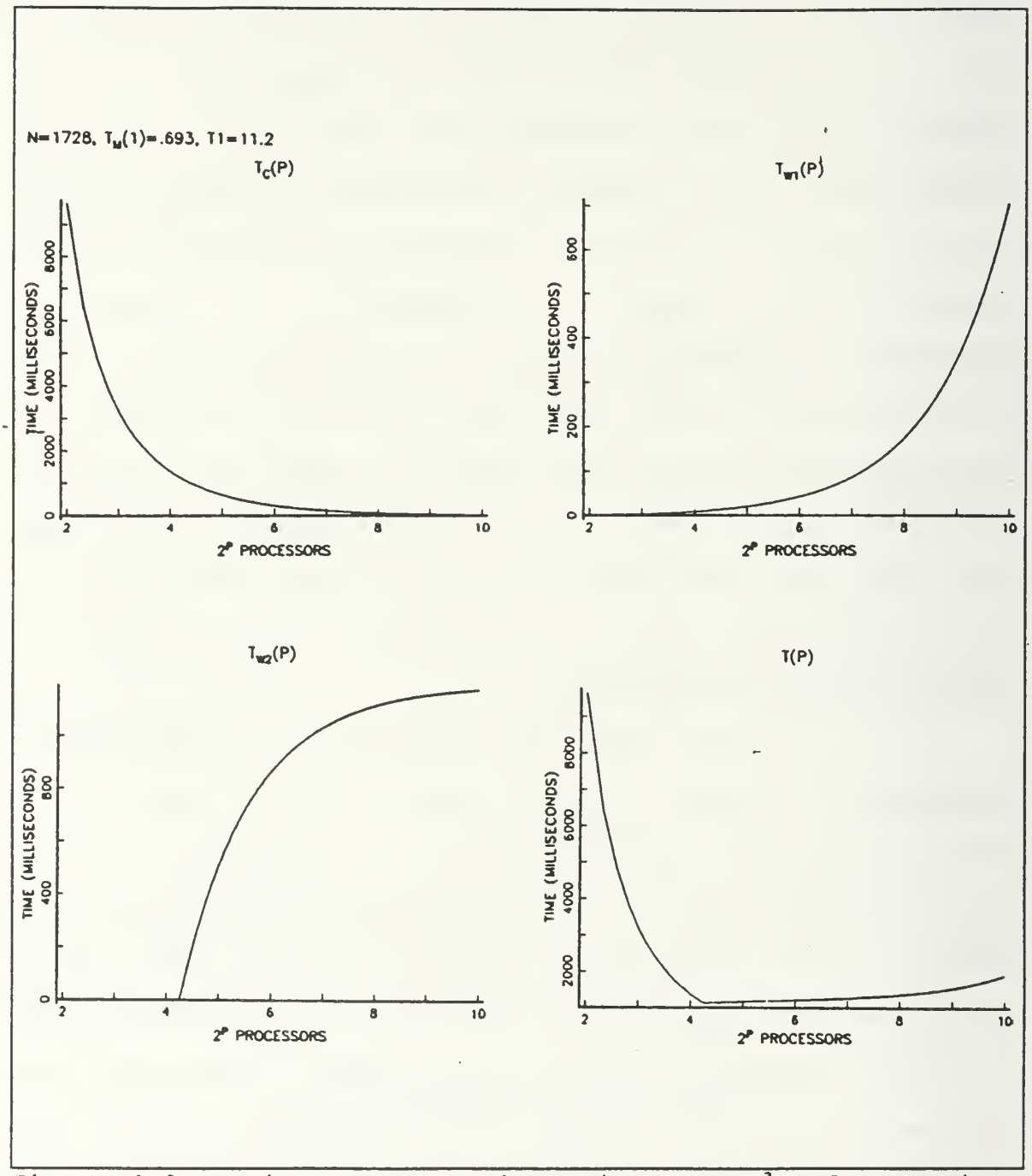

Figure 4.9 Estimated Execution Time of P^3T for Various Hypercube Sizes

algorithm could be expected. For further information on the INTEL Concurrent File System see (iPSC/2 User's Guide, 1990, 7-1 - 7-18).

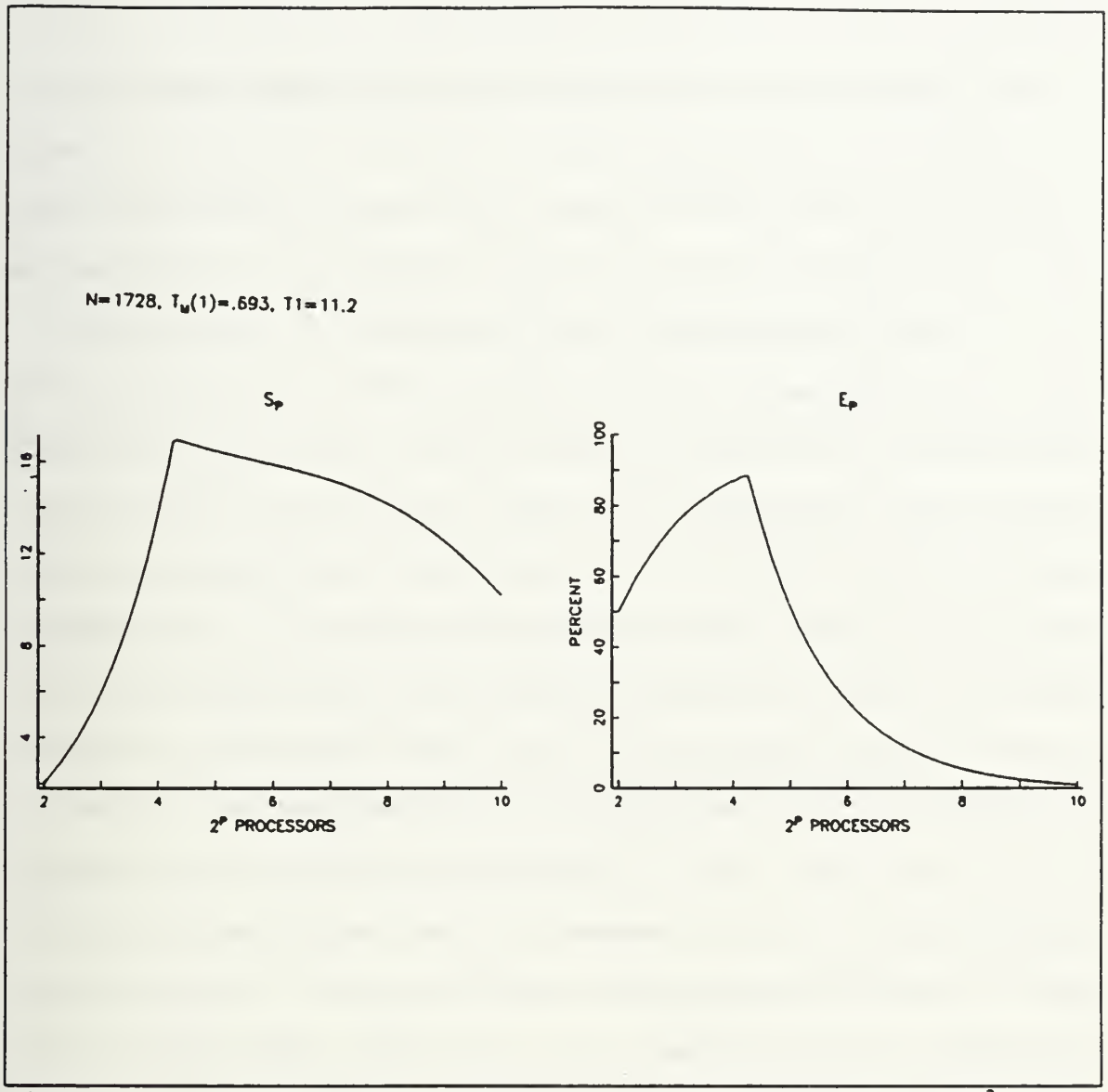

Figure 4.10 Estimated Speedup and Efficiency of P^3T for Various Hypercube Sizes

V. CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

The ultimate objective of this thesis is determine the parallel computing potential of the NAVSPASUR satellite motion model. From the results given in Chapter IV, vectorization and control decomposition strategies proved not beneficial in significantly reducing the computation time of the NAVSPASUR model. Very few apparent vector operations exist in the model, and any attempt to transform the formulas into vector operations became algebraically overwhelming. Although the analytic formulas of the NAVSPASUR model are quite lengthy and complex, they proved to be not computationally intensive enough to warrant decomposition of the algorithm by tasks.

On the other hand, the domain decomposition strategy showed promise if the satellites are propagated in a batch mode. The P^3T algorithm was simple to apply. The algorithm provided the flexibility to vary the dimension of the hypercube and to easily modify the model itself. Although only a maximum efficiency of .67 was achieved, the potential efficiency was artificially bounded by the number of nodes available with the specific hypercube multi-computer used. Having a maximum of only eight nodes available, the efficiency of P^3T is bounded above by .75. Using the model of P^3T's total execution time described in Chapter IV, it was shown that a

maximum efficiency of just under 90 percent could be achieved with a hypercube consisting of 16 nodes. The corresponding speedup factor of nearly 14 would significantly reduce the time to predict the state vectors for several thousand satellites.

The success of **P^{3}T** manifests several possible areas of future research. One area would be to apply **P^{3}T** to higher dimension cubes and validate the estimates of speedup and efficiency using more than eight nodes. Because the number of working nodes is not fixed for this algorithm, **P^{3}T** could be applied easily to any size hypercube with no modifications. Once the optimal size is found, one could attach several sub-cubes of the optimal dimension and determine the benefit of propagating several smaller catalogs of satellite data instead of propagating one large catalog.

Another possible area of research would be to modify the current satellite motion model to increase the accuracy of its predictions. The results in the previous chapter showed an increase in performance of the **P^{3}T** if the amount of computation was increased. Hence, greater accuracy could be achieved in far less time using the **P^{3}T** algorithm than the time using the original serial algorithm. Additionally, from these results, the parallel computing potential of satellite motion models that are more computationally intensive would be greater. For example, semi-analytic models which combine the

benefits of analytic and numerical models might be good candidates.

Overall, the main lesson learned from this thesis is that satellite position prediction can be made more timely through parallel computing. Although the best method of parallelization might vary depending on the specific model used, parallel computing is a viable option to achieve timely satellite position prediction for the growing number of objects in orbit around the Earth.

APPENDIX A

INPUT/OUTPUT OF PPT2

This appendix contains a listing of the primary variables used by NAVSPASUR subroutine, **PPT2**. Tables A.1 - A.4 are extracted from (Solomon, 1991, pp. 12 - 14). Table A.5 was interpreted from the NAVSPASUR source code.

Table A.1 PPT2 Calling List

Variable	Definition	Input	Output
IND	control variable	I	
TM	time	I	O
KZ	0 for inertial coordinates 1 for Earth-fixed	I	

Table A.2 PPT2 Input Control Variables

Variable		Option
KF(1)	IND	
0	<0	Position, no partial derivatives
0	0	Position, partial derivatives
0	1	Secular recovery
0	2	Epoch update
KZ		
0		Inertial coordinates
1		Earth-fixed coordinates
KF(3)		
0		Secular and drag corrections only
1		Secular, drag, and periodic correctio
>1		Secular, drag, periodic, and sectoral/tesseral corrections
KF(2), KF(4)-KF(10)		Not used by PPT2

Table A.3 PPT2 Common Blocks

Block	Variable	Description	Input	Output
CONS	A(64)	constants set by subroutine CONS1	I	
PPT	F(25)	stored mean elements	I	O
	OSC(10)	osculating elements		O
	KF(10)	control variables	I	O
	CF(10)	used for appearance prediction	I	O
	BS(3,4)	observation stations position $(R, \hat{z}, \hat{e}, \hat{n})$	I	
	U(3)	$\hat{r}$		O
	V(3)	$\hat{v}$		O
	W(3)	$\vec{w}$		O
	R	r		O
	VEL(3)	$\dot{r}$, velocity		O
DCSUB	PE(6,8)	partial derivatives		O
	. . .	others not used in PPT2		
FOREO	RHO(3)	$\vec{\rho}$		O
	RHOS	ρ^2		O
	HDR	$\hat{z} \cdot \vec{\rho}$		O
	HDV	$\hat{z} \cdot \dot{r}$		O
	RDV	$\vec{\rho} \cdot \dot{r}$		O
	DEL	Δf		O
	ITER	iteration counter		O

Table A.4 Array F

Index	Variable	Definition	Input	Output
1	l"	mean mean anomaly	I	O
2	m	mean motion	I	O
3	M_2	first decay term	I	O
4	M_3	second decay term	I	O
5	e"	eccentricity	I	O
6	g"	mean argument of perigee	I	O
7	h"	mean ascending node	I	O
8	cos I"	cosine inclination	I	O
9	ė	epoch	I	O
10	#	revolution number	I	O
11	dg"/dl"			O
12	dh"/dl"			O
13	a"	mean semi-major axis		O
14	ȧ			O
15	sin I"	sine inclination		O
16	ė			O
17-25		not used in PPT2		

Table A.5 Array OSC

Index	Variable	Definition
1	cos f	cosine true anomaly
2	sin f	sine true anomaly
3	l	osculating mean anomaly
4	e	osculating eccentricity
5	g	osculating argument of perigee
6	h	osculating ascending node
7	cos I	cosine osculating inclination
8	a	osculating semi-major axis
9	Δl	secular and drag correction term for l
10		not used by PPT2

APPENDIX B

INTEL iPSC/2 SPECIFICATIONS

This appendix contains a summary of the iPSC/2 hypercube multi-computer specifications as described in (iPSC/2 User's Guide, 1990, pp. 1-1 - 1-11). The exact performance values were obtained from (Arshi, 1988, pp. 17 - 22).

iPSC/2

System Resource Manager (Host)

Central Processing Unit	INTEL 80386 (4 MIP)
Numeric Processing Unit	INTEL 80387 (250 KFLOP 64-bit)
Memory	8 Mbyte
External Communication	Ethernet TCP/IP local area network port
Operating System	AT&T UNIX, Version V, Release 3.0

Nodes

Node Processor	INTEL 80386 (4 MIP)
Numeric Co-processor	INTEL 80387 (250 KFLOP 64-bit)
Operating System	NX/2

Internal Communication

Direct-Connect™ Routing	Message Latency -- 350 µsec
	Message Bandwidth -- 2.8 Mbytes/sec

APPENDIX C

P^3T-4 SOURCE CODE LISTING

This appendix contains the listing of host and node programs comprising P^3T-4 program set. The host program reads the satellite data, loads the node program, and writes the results to an external output file. The node program contains the instructions for the four nodes to complete their respective portions of the P^3T-4 parallel algorithm. The node program is limited to only satellite state vector prediction.

Host Program:

```
      program P3T4h
*
*   This host program reads the satellite data from a
*   external file, loads the program P3T-4n on the hypercube
*   nodes, and writes results to external output file.
*
      implicit real*8 (a-h,o-z)

      dimension sat(49,6000),end(49,1)
      integer pid,msglen,eof

      data pid/0/,msglen/392/
      data isat/1/

      call setpid(pid)

*   Load node program P3T4n on nodes

      print*,'host loading nodes'
      call load('p3t4n',-1,pid)

*   Read satellite data and send data to nodes until reach
*   end-of-file

      open(unit=10,file='/usr/phipps/in4',form='unformatted')
      read (unit=10,iostat=eof) (sat(j,1),j=1,49)

   20 if (eof.ge.0) then
         call csend(5,sat(1,isat),msglen,-1,pid)
         isat=isat+1
         read (unit=10,iostat=eof) (sat(j,isat),j=1,49)

*   Receive results from nodes
```

```
            call crecv(99,sat(1,isat-1),msglen)
          go to 20
          endif

*    Send message for nodes to halt and clear cube for next process

          call csend(5,end,msglen,-1,pid)
          close(unit=10)
          call killcube (-1,-1)

*    Write results to external file

          open(unit=11,file='/usr/phipps/out4',form='unformatted')
          do 22 i=1,isat-1
    22    write(11,*)(sat(j,i),j=1,49)
          close(unit=11)

          stop
          end
```

Node Program:

```
        program P3T4n
*     This node program is a parallel code for satellite
*     position prediction using the NAVSPASUR model.  This
*     program employs four nodes using the control
*     decomposition strategy.  The specific tasks for each
*     node are separated by logical if statements.

        implicit real*8 (a-h,m-z)
        real*8 kz

*   Declare cube specific function and variables as integers

        integer mynode,mclock,msgwait,myhost
        integer mynod,pid,msglen,hostid

        common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
        common /cons/c(25)
        common /n/theta2,eta0,eta20,esq0
        common /g/g2,g2p,g3,g4,g5
        common /crit/t2
        common /sec/agda,agde,agdi,agdl,agdg,agdh

        data pid/0/,msglen/8/

        hostid=myhost()
        mynod=mynode()

*   Synchronize nodes

        call gsync()

*   Nodes set constants and receive data from host

          call cons1
          call crecv(5,f,msglen*49)

*   Nodes continue to execute ppt3 until catalog of
*   satellite data is exhausted

 1100   if(tm.eq.0.0d0) go to 1101

*   Node 2 computes new T2 and sends to other nodes

          if(mynod.eq.2)then
             call critincl
             call csend(mynod,t2,msglen,-1,pid)
          endif

*   Nodes execute respective portion of subroutine ppt3

          call ppt3(mynod)

*   Node 0 send results to host

          if(mynod.eq.0)then
             call csend(99,f,msglen*49,hostid,pid)
          endif
```

```
*   Repeat until catalog of satellite data is exhausted

          call crecv(5,f,msglen*49)
        go to 1100

 1101   continue

*   End main program

        stop
        end

*************************************************************

        subroutine ppt3(mynod)

*   This subroutine preschedules the tasks for each node
*   numbered 0 through 3.  Tasks are partitioned by logical
*   if statements

*    Declare variables

        implicit real*8 (a-h,m-z)
        real*8 kz

*   Declare cube specific function and variables as integers

        integer mynode,mclock,msgwait
        integer mynod,pid,msglen
        integer msg4(10)

        dimension msg1(2),msg3(2)

*   Define common blocks

        common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
        common /cons/c(25)
        common /n/theta2,eta0,eta20,esq0
        common /g/g2,g2p,g3,g4,g5
        common /crit/t2
        common /sec/agda,agde,agdi,agdl,agdg,agdh

        equivalence (msg1(1),osc(6)),(msg3(1),osc(1))


        data pid/0/,msglen/8/



        if(mynod.ne.2)then
          esq0=f(5)*f(5)
          theta2=f(8)*f(8)
          eta20=1.0d0-esq0
          eta0=dsqrt(eta20)

*0000000000000000000000000000000000000000000000000000000000
        if(mynod.eq.0)then


*   Post asynchronous receive message commands
```

```
      msg4(1)=irecv(2,t2,msglen)
      msg4(2)=irecv(1,osc(5),msglen)
      msg4(3)=irecv(3,agda,msglen*6)

*     Recover a from mean orbital elements

      call recover

*     Send a to all nodes

      call csend(mynod,f(13),msglen,-1,pid)

*     Compute g2,g2p,g3,g4,g5

      call gamma
      h1=tm-f(9)

*     Compute osculating l,e,a

      osc(9)=((f(4)*h1+f(3))*h1+f(2))*h1
      osc(3)=pie(osc(9)+f(1))
      f(14)=-4.0d0*f(13)*f(3)/(f(2)*3.0d0)
      f(16)=f(5)*eta20*f(14)/f(13)
      osc(4)=dmin1(dmax1(0.0d0,f(5)+f(16)*h1),.99999999d0)
      osc(8)=dmax1(1.0d0,f(13)+f(14)*h1)

*     Send l, e and a to other nodes

      call message(osc(3),osc(4),osc(8),mynod,-1,0)


*     Compute sin I

      f(15)=dsqrt(1.0d0-theta2)

*     Receive t2 from node 2

      call msgwait(msg4(1))
      msg4(4)=irecv(2,msg3,2*msglen)

*     Make preliminary computations for ed11

      tt2=theta2*t2
      p1=(-8.0d0*tt2-3.0d0)*theta2+1.0d0
      p2=(-40.d0*tt2-11.0d0)*theta2+1.0d0
      vle1=0.125d0*g2p*p2-(5.0d0*g4*p1/(12.0d0*g2p))
      vle1=vle1*f(5)*eta20
      p1=(-24.0d0*tt2-9.0d0)*theta2+1.0d0
      p2=(1.25d0+.9375d0*esq0)*g5
      vle2=(p1*p2+g3)*eta20*f(15)/(4.0d0*g2p)
      vll2=vle2+3.0d0*eta20*0.15625d0*esq0*f(15)*g5*p1/g2p
      p1=(-16.0d0*tt2-5.0d0)*theta2+1.0d0
      vle3=p1*f(15)*eta20*esq0*g5*35.0d0/(-3.84d02*g2p)

*     Receive g from node 1

      call msgwait(msg4(2))
      msg4(5)=irecv(1,DE,msglen)

*     Compute ed11
```

```
      cosg=dcos(osc(5))
      sing=dsin(osc(5))


      p1=4.0d0*cosg**3-3.0d0*cosg
      p2=2.0d0*sing*cosg
      ed1l=(vle1*p2-vll2*cosg-vle3*p1)*eta0

*     Receive cos f, sin f from node 2

      call msgwait(msg4(4))
      msg4(6)=irecv(2,OS,msglen)

*     Compute ed2l

      w22=(1.0d0+osc(4)*osc(1))*(2+osc(4)*osc(1))/eta20
      p1=(3.0d0-4.0d0*osc(2)**2)*osc(2)*(1.0d0-2.0d0*sing**2)
      p1=(4.0d0*osc(1)**2-3.0d0)*osc(1)*p2+p1
      p3=(1.0d0-2.0d0*sing**2)*osc(2)+osc(1)*p2
      p4=3.0d0*(1.0d0-theta2)
      p5=-1.0d0+3.0d0*theta2
      eta30=eta20*eta0
      ed2l=(p1*(w22+1.0d0/3.0d0)+p3*(1.0d0-w22))*p4
      ed2l=(ed2l+osc(2)*(w22+1.0d0)*p5*2.0d0)*eta30*g2p/-4.0d0

*     Send ed2l to node 2

      call csend(mynod,ed2l,msglen,2,pid)

*     Compute ed1 and a

      ed1=ed1l+ed2l
      p1=(cosg*cosg-sing*sing)*(osc(1)*osc(1)-osc(2)*osc(2))-
     &     p2*2.0d0*osc(1)*osc(2)
      p6=(1.0d0+osc(4)*osc(1))**3
      osc(8)=osc(8)*(1.0d0+g2p/eta20*(p5*(p6-eta30)+
     &       p4*p6*p1))


*     Receive sectoral corrections from node 3

      call msgwait(msg4(3))
      DL=ed1+agdl
      osc(8)=osc(8)+agda

*     Receive DE from node 1

      call msgwait(msg4(5))
      msg4(7)=irecv(1,msg1,msglen*2)

*     Compute final value for e and l

      esq=de**2+dl**2
      osc(4)=dsqrt(esq)
      sinl=dsin(osc(3))
      cosl=dcos(osc(3))
      osc(3)=artnq(DE*sinl+DL*cosl,DE*cosl-DL*sinl)

*     Solve Kepler's eqn

      call kepler
```

```
            if(kz.ne.0.0d0)cf=c(12)
            eta2=1.0d0-esq
            r=osc(8)*eta2/(1.0d0+osc(4)*osc(1))

*     Receive final computations from other processors

            call msgwait(msg4(7))
            call msgwait(msg4(6))

*     Compute g

            osc(5)=os-osc(3)-osc(6)

*     Compute position and velocity vectors

            cosg=dcos(osc(5))
            sing=dsin(osc(5))
            cosh=dcos(osc(6))
            sinh=dsin(osc(6))

            p1=sing*osc(1)+cosg*osc(2)
            p2=cosg*osc(1)-sing*osc(2)
            sini=dsqrt(1.0d0-osc(7)**2)

            u(1)=cosh*p2-sinh*p1*osc(7)
            u(2)=sinh*p2+cosh*p1*osc(7)
            u(3)=p1*sini


            v(1)=-cosh*p1-sinh*p2*osc(7)
            v(2)=-sinh*p1+cosh*p2*osc(7)
            v(3)=p2*sini

            w(1)=sinh*sini
            w(2)=-cosh*sini
            w(3)=osc(7)

            p3=osc(4)*osc(2)
            p4=osc(4)*osc(1)+1.0d0
            p5=dsqrt(eta2*osc(8))

            do 11 i=1,3
   11       vel(i)=(p3*u(i)+p4*v(i))/p5
            vel(1)=vel(1)+cf*r*u(2)
            vel(2)=vel(2)-cf*r*u(1)

*     End for node 0


            endif

*0000000000000000000000000000000000000000000000000000000000000000
*1111111111111111111111111111111111111111111111111111111111111111

*     Begin node 1

            if(mynod.eq.1)then


            msg4(1)=irecv(0,f(13),msglen)
            msg4(2)=irecv(2,t2,msglen)
```

```
      msg4(3)=irecv(3,adga,msglen*6)

*     Compute sin i and tan i

      f(15)=dsqrt(1.0d0-theta2)
      tani=f(15)/f(8)
      h1=tm-f(9)
      p1=1.0d0-5.0d0*theta2
      p2=-35.0d0+24.0d0*eta0+25.0d0*eta20
      p3=90.0d0-192.0d0*eta0-126.0d0*eta20
      p4=385.0d0+360.0d0*eta0+45.0d0*eta20
      p5=-270.0d0+126.0d0*eta20
      p6=385.0d0-189.0d0*eta20
      theta4=theta2*theta2

*     Receive a from node 0

      call msgwait(msg4(1))

*     Compute g2,g2p,g3,g4,g5

      call gamma

*     Compute g
      osc(9)=((f(4)*h1+f(3))*h1+f(2))*h1
      f(11)=-1.5d0*g2p*p1+.09375d0*g2p**2*(p2+p3*theta2
     &      +p4*theta4)+.3125d0*g4*(21.0d0-9.0d0*eta20+p5*theta2
     &      +p6*theta4)
      q1=f(2)*f(13)**1.5d0
      f(11)=f(11)/q1
      osc(5)=pie(f(6)+f(11)*osc(9))

*     Send g to all nodes

      call csend(mynod,osc(5),msglen,-1,pid)

*     Complete preliminary computations to solve for d1e and d1i

      cosg=dcos(osc(5))
      sing=dsin(osc(5))
      p3=(3.0d0-4.0d0*sing**2)*sing
      p4=2.0d0*cosg**2-1.0d0
      p5=2.0d0*sing*cosg
      p6=cosg*cosg-sing*sing

*     Receive t2 from node 2

      call msgwait(msg4(2))
      msg4(4)=irecv(2,msg3,msglen*2)

      tt2=theta2*t2
      p1=(-8.0d0*tt2-3.0d0)*theta2+1.0d0
      p2=(-40.d0*tt2-11.0d0)*theta2+1.0d0
      vle1=0.125d0*g2p*p2-(5.0d0*g4*p1/(12.0d0*g2p))
      vle1=vle1*f(5)*eta20
      p1=(-24.0d0*tt2-9.0d0)*theta2+1.0d0
      p2=(1.25d0+.9375d0*esq0)*g5
      vle2=(p1*p2+g3)*eta20*f(15)/(4.0d0*g2p)
      p1=(-16.0d0*tt2-5.0d0)*theta2+1.0d0
      vle3=p1*f(15)*eta20*esq0*g5*35.0d0/(-3.84d02*g2p)
```

```
*   Compute d1e and d1i

        d1e=vle1*p4+vle2*sing+vle3*p3
        d1i=-f(5)*d1e/(eta20*tani)

*   Receive l, e and a from node 0

        call message(osc(3),osc(4),osc(8),mynod,0,1)

*   Receive cos f, sin f and r from node 2

        call msgwait(msg4(4))

*   Compute d2i

        p1=(4.0d0*osc(1)**2-3.0d0)*osc(1)*p4
        p1=p1-p5*(3.0d0-4.0d0*osc(2)**2)*osc(2)
        p2=p4*osc(1)-p5*osc(2)
        p3=(osc(1)*osc(1)-osc(2)*osc(2))*p6-2.0d0*p5*osc(1)*osc(2)
        d2i=((p1+3.0d0*p2)*f(5)+3.0d0*p3)*f(15)*f(8)*g2p

*   Compute d2e

        w20=osc(1)*(3.0d0+osc(4)*osc(1)*(3.0d0+osc(4)*osc(1)))
        p4=eta0+1.0d0/(1.0d0+eta0)
        p5=1.0d0-theta2

        d2e=0.5d0*g2p*((3.0d0*theta2-1.0d0)*(w20+f(5)*p4)
     &      +3.0d0*p5*(w20+f(5))*p3-eta20*p5*(3.0d0*p2+p1))

*   Compute di and de

        di=d1i+d2i
        de=d1e+d2e

*   Receive sectoral corrections

        call msgwait(msg4(3))

*   Compute DE and send to node 0

        DE=de+osc(4)+agde
        call csend(mynod,DE,msglen,0,pid)

*   Compute DI

        p1=(f(8)+1.0d0)/2.0d0
        p2=dsqrt(p1)
        p3=dsqrt(1.0d0-p1)
        DI=p3+.5d0*p2*(di+agdi)

*   Receive osc(6) and DH from node 3

        call message(osc(6),DH,dummy,mynod,3,1)

*   Compute final cos i and h and send to node 0

        osc(7)=1.0d0-2.0d0*(DI**2+DH**2)
        sinh=dsin(osc(6))
        cosh=dcos(osc(6))
        osc(6)=artnq(DI*sinh+DH*cosh,DI*cosh-DH*sinh)
```

```
          call csend(mynod,msg1,msglen*2,0,pid)

*    End of node 1

          endif

*11111111111111111111111111111111111111111111111111111111
*33333333333333333333333333333333333333333333333333333333

*    Begin node 3

          if(mynod.eq.3)then

          msg4(1)=irecv(0,f(13),msglen)
          msg4(2)=irecv(2,t2,msglen)
          msg4(3)=irecv(1,osc(5),msglen)

          sini2=1.0d0-theta2
          f(15)=dsqrt(sini2)
          h1=tm-f(9)
          osc(9)=h1*(f(2)+h1*(f(3)+h1*f(4)))
          theta4=theta2*theta2
          p1=1.0d0-5.0d0*theta2
          p2=-35.0d0+24.0d0*eta0+25.0d0*eta20
          p3=90.0d0-192.0d0*eta0-126.0d0*eta20
          p4=385.0d0+360.0d0*eta0+45.0d0*eta20
          p5=-270.0d0+126.0d0*eta20
          p6=385.0d0-189.0d0*eta20
          p7=-5.0d0+12.0d0*eta0+9.0d0*eta20
          p8=35.0d0+36.0d0*eta0+5.0d0*eta20
          p9=3.0d0-7.0d0*theta2

*    Receive a from node 0

          call msgwait(msg4(1))

*    Compute g2,g2p,g3,g4,g5

          call gamma

*    Compute h

          f(12)=(-3.0d0*g2p+.375d0*g2p**2
     &         *(p7-p8*theta2)+1.25d0*g4*(5.0d0-3.0d0*eta20)*p9)*f(8)
          q1=f(2)*f(13)**1.5d0
          f(12)=f(12)/q1
          if(kz.ne.0.0d0)cf=c(12)

          osc(6)=pie(f(7)+f(12)*osc(9))-cf*(tm-c(1))

*    Send h to node 2

          call csend(mynod,osc(6),msglen,2,pid)

*    Perform preliminary computations for sectoral corrections

          f(11)=-1.5d0*g2p*p1+.09375d0*g2p**2*(p2+p3*theta2
     &         +p4*theta4)+.3125d0*g4*(21.0d0-9.0d0*eta20+p5*theta2
     &         +p6*theta4)
          f(11)=f(11)/q1
          agda=10.0d0
```

```
      call sector(h1)

*     Receive t2 from node 2

      call msgwait(msg4(2))
      msg4(4)=irecv(2,msg3,msglen*2)

*     Complete preliminary computations to compute sini d1h

      tt2=theta2*t2
      p1=(40.0d0*tt2+16.0d0)*tt2+3.0d0
      p2=(2.0d02*tt2+80.0d0)*tt2+11.0d0
      vlh1i=(5.0d0*g4*p1/(12.0d0*g2p)-.125*g2p*p2)*f(15)*esq0*f(8)

      p2=4.0d0+3.0d0*esq0
      p3=(-24.0d0*tt2-9.0d0)*theta2+1.0d0
      vlh2i=((f(5)*f(8)*.25d0)/g2p)*(g3+(5.0d0*g5/16.0d0)*p2*p3
     &+15.0d0*g5*(f(15)**2)*p2*p1/8.0d0)

      p2=(-8.0d0*tt2-2.5d0)*theta2+0.5d0
      p3=2.0d0*p1-1.0d0
      vlh3i=(-35.0d0*g5*esq0*f(5)*f(8))*(p2+p3*f(15)*f(15))
     &/(576.0d0*g2p)

*     Receive l, e and a from node 0

      call message(osc(3),osc(4),osc(8),mynod,0,1)

*     Receive g from node 1

      call msgwait(msg4(3))

*     Compute sectoral corrections and send to all nodes

      agda=0.0d0
      agde=0.0d0
      agdi=0.0d0
      agdl=0.0d0
      agdg=0.0d0
      agdh=0.0d0

      call sector(h1)

      call csend(mynod,agda,msglen*6,-1,pid)

*     Compute sini d1h

      cosg=dcos(osc(5))
      sing=dsin(osc(5))
      p1=2.0d0*sing*cosg
      p2=(4.0d0*cosg**2-3.0d0)*cosg
      sinid1h=vlh1i*p1+vlh2i*cosg+vlh3i*p2

*     Receive cosf, sinf from node 2

      call msgwait(msg4(4))

*     Compute sini d2h

      w17=artnq(osc(2),osc(1))+osc(4)*osc(2)-pie(osc(3))
      p2=2.0d0*cosg**2-1.0d0
```

```
      p3=(4.0d0*osc(1)**2-3.0d0)*osc(1)
      p4=(3.0d0-4.0d0*osc(2)**2)*osc(2)
      p5=p1*(6.0d0*osc(1)**2-3.0d0)+6.0d0*p2*osc(1)*osc(2)
      p6=(p1*osc(1)+p2*osc(2))*3.0d0
      w21=p5+(p6+p1*p3+p2*p4)*osc(4)
      sinid2h=-0.5d0*g2p*f(8)*f(15)*(6.0d0*w17-w21)


*     Compute DH and send osc(6) and DH to node 1

      DH=0.5d0*(sinid1h+sinid2h+agdh)/dsqrt(0.5d0+0.5d0*f(8))
      call message(osc(6),DH,0.0d0,mynod,1,0)

*     End of node 3

      endif


*3333333333333333333333333333333333333333333333333333333333

      else

*2222222222222222222222222222222222222222222222222222222222

*     Begin node 2


      msg4(1)=irecv(0,f(13),msglen)
      msg4(2)=irecv(1,osc(5),msglen)
      msg4(3)=irecv(3,osc(6),msglen)

*     Perform preliminary computations for d1z

      f(15)=dsqrt(1.0d0-theta2)
      tt2=theta2*t2
      esq0=f(5)*f(5)
      eta20=1.0d0-esq0
      eta0=dsqrt(eta20)
      eta30=eta20*eta0

*     Receive a from node 0

      call msgwait(msg4(1))

*     Compute g2,g2p,g3,g4 and g5

      call gamma

*     Compute d1z

      p1=(eta30-1.0d0)*.125d0
      p2=((-40.0d0*tt2-11.0d0)*theta2+1.0d0)*g2p
      p3=10.0d0*g4/(3.0d0*g2p)
      p4=((-8.0d0*tt2-3.0d0)*theta2+1)*p3
      p5=(20.0d0*tt2+8.0d0)*tt2+1.0d0
      p6=(10.0d0*p5+1.0d0)*g2p
      p7=(2.0d0*p5+1.0d0)*p3
      p8=25.0d0*esq0*tt2*tt2*theta2*(g2p-.2d0*p3)
      p9=((-2.0d02*tt2-33.0d0)*theta2+1.0d0)*g2p
      p10=((-40.0d0*tt2-9.0d0)*theta2+1.0d0)*p3
      vls1=p1*(p2-p4)-.125d0*esq0*f(8)*(p6-p7)
```

```
     &       +p8-.0625d0*esq0*(p9-p10)

          p1=eta0+1.0d0/(1.0d0+eta0)
          p2=f(8)/(1.0d0+f(8))
          p3=4.0d0+3.0d0*esq0
          p4=(-24.0d0*tt2-9.0d0)*theta2+1.0d0
          p5=(esq0-eta30)*3.0d0+11.0d0
          p6=(20.0d0*tt2+8.0d0)*tt2+1.0d0
          p7=(p1+p2)/4.0d0
          p8=(g3+.3125d0*p3*g5*p4)/g2p
          p9=p5*.15625d0*g5*p4/g2p
          p10=(1.0d0-f(8))*.46875d0*(p6*2.0d0+1.0d0)*p3*g5*f(8)/g2p
          vls2=(p7*p8+p9+p10)*f(5)*f(15)

          p7=3.0d0*eta30-3.0d0-(2.0d0+p2)*esq0
          p8=(-16.0d0*tt2-5.0d0)*theta2+1.0d0
          p9=(1.0d0-f(8))*f(8)*.060763884d0*esq0*(1.0d0+4.0d0*p6)
          vls3=(p7*.030381944d0*p8-p9)*f(15)*f(5)*g5/g2p


*    Receive g from node 1 and compute d1z

          call msgwait(msg4(2))

          sing=dsin(osc(5))
          cosg=dcos(osc(5))
          p11=2.0d0*sing*cosg
          p12=2.0d0*cosg**2-1.0d0
          p13=(cosg*cosg-sing*sing)*cosg-2.0d0*cosg*sing*sing

          d1z=vls1*p11+vls2*cosg+vls3*p13

*    Receive l, e, and a from node 0

          call message(osc(3),osc(4),osc(8),mynod,0,1)
          msg4(4)=irecv(0,ed2l,msglen)

*    Solve Kepler's equation and send cosf, sinf to all nodes

          call kepler
          call csend(mynod,msg3,msglen*2,-1,pid)

*    Complete preliminary computations for d2z

          w17=artnq(osc(2),osc(1))+osc(4)*osc(2)-pie(osc(3))
          p3=(4.0d0*osc(1)**2-3.0d0)*osc(1)
          p4=(3.0d0-4.0d0*osc(2)**2)*osc(2)
          p5=p11*(6.0d0*osc(1)**2-3.0d0)+6.0d0*p12*osc(1)*osc(2)
          p6=(p11*osc(1)+p12*osc(2))*3.0d0
          w21=p5+(p6+p11*p3+p12*p4)*osc(4)
          p7=(-5.0d0*theta2+2.0d0*f(8)+3.0d0)*w21
          p8=(-5.0d0*theta2+2*f(8)+1.0d0)*w17

*    Receive ed2l from node 0

          call msgwait(msg4(4))

*    Compute d2z

          d2z=-f(5)*ed2l*(p1-1.0d0)/eta30-(6.0d0*p8-p7)*g2p*0.25d0
```

```
*  Receive h form node 3

         call msgwait(msg4(3))

*   Receive sectoral corrections

         call crecv(3,agda,msglen*6)

*   Compute OS

         OS=osc(3)+osc(5)+osc(6)+d1z+d2z+agdg


*   Send OS to node 0

         call csend(mynod,OS,msglen,0,pid)

*   End of node 2


*22222222222222222222222222222222222222222222222222222222

         endif

*   Return to main program

         return
         end
*************************************************************
         function pie(x)

*   This double precision function computes value of
*   x mod 2*pi

         implicit real*8 (a-h,m-z)

         pi2=6.2831853071d0
         pie=dmod(x,pi2)
         if(pie)90,91,91
   90    pie=pie+pi2

   91    return
         end
*************************************************************
         function artnq(t1,t2)

*   This double precision function computes inverse tangent of t1/t2
*   for range 0 to 2*pi

         implicit real*8 (a-h,m-z)

         if(dabs(t1)-dabs(t2))100,104,104
  100    artnq=datan(t1/t2)

         if(t2)101,102,102
  101    artnq=artnq+3.14159265358979d0
         go to 105

  102    if(t1)103,105,105
  103    artnq=artnq+6.28318530717959d0
         go to 105
```

```
  104     artnq=atan(-t2/t1)+1.57079632679489d0

          if(t1)101,105,105
  105     return
          end
*************************************************************
          subroutine recover

*   This subroutine recovers the value of a"
*   iteratively from m

          implicit real*8 (a-h,m-z)
          real*8 kz

          common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
          common /cons/c(25)
          common /n/theta2,eta,eta2,esq


          f(13)=f(2)**(-2.0d0/3.0d0)


          do 110 i=1,5
             g2p=c(3)/(f(13)*eta2)**2
             g4=c(5)/(f(13)*eta2)**4
             p1=((35.0d0*theta2)-30.0d0)*theta2+3.0d0
             p2=25.0d0*eta2+144.0d0*eta+105.0d0
             p3=-90.0d0*eta2-96.0d0*eta+30.0d0
             p4=25.0d0*eta+16.0d0*eta-15.0d0
             p5=3.0d0*theta2-1.0d0

  110    f(13)=((1.0d0+1.5d0*g2p*eta*p5+.09375d0*g2p**2
     &         *(p4+theta2*(p3+p2*theta2))+.9375d0*g4*eta*esq*p1)
     &         /f(2))**(2.0d0/3.0d0)

          return
          end
*************************************************************
          subroutine gamma

*   This subroutine computes the dimensionless quantities

          implicit real*8 (a-h,m-z)
          real*8 kz

          common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
          common /cons/c(25)
          common /n/theta2,eta0,eta20,esq0
          common /g/g2,g2p,g3,g4,g5

          g2=c(3)/f(13)**2
          g2p=g2/eta20**2
          g3=c(4)/(f(13)*eta20)**3
          g4=c(5)/(f(13)*eta20)**4
          g5=c(6)/(f(13)*eta20)**5

          return
          end
*************************************************************
```

```
      subroutine critincl

*   This subroutine computes the value of T2

      implicit real*8 (a-h,m-z)
      real*8 kz

      common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
      common /n/theta2,eta0,eta20,esq0
      common /crit/t2

      theta2=f(8)*f(8)
      t1=1.0d0-5.0d0*theta2
      beta=1.0d02/(2.0d0**11)

      p3=beta*t1*t1
      p1=dexp(-p3)
      p2=1.0d0+p1
      p1ex=p1*p1
      p4=1.0d0
      p3ex=-0.5d0*p3

      do 210 i=2,13
         if(i.le.11)p2=p2*(1.0d0+p1ex)
         p1ex=p1ex*p1ex
         p4=p4+p3ex
 210  p3ex=-p3*p3ex/dble(i+1)

      t2=p2*p4*beta*t1

      return
      end
*************************************************************
      subroutine kepler

*   This subroutine solves Kepler's Equation using Steffenson's
*   Method and then computes cos f and sin f

      implicit real*8 (a-h,m-z)
      real*8 kz

      common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz

      e3=osc(3)
      do 410 i=1,20
         e1=e3
         e3=osc(3)+osc(4)*dsin(e3)
         if(dabs(e3-e1).lt.1.0d-08) go to 420
         e2=e3
         e3=osc(3)+osc(4)*dsin(e3)
         if(dabs(e3-e2).lt.1.0d-08) go to 420
 410  e3=e3+(e3-e2)**2/(2.0d0*e2-e1-e3)

 420  cosf=dcos(e3)
      e1=1.0d0-osc(4)*cosf
      osc(1)=(cosf-osc(4))/e1
      eta=dsqrt(1.0d0-osc(4)**2)
      osc(2)=(eta*dsin(e3))/e1

      return
      end
```

```
*************************************************************
          subroutine sector(h1)

*    This subroutine computes sectoral correction terms to be added
*    to the variation of each of the orbital elements

          implicit real*8 (a-h,m-z)
          real*8 kz
          dimension clm(8),slm(8),tf(8),tfp(8),te(8,8)

          common /ppt/f(25),osc(9),u(3),v(3),w(3),vel(3),r,tm,kz
          common /cons/c(25)
          common /n/theta2,eta,eta2,esq
          common /sec/agda,agde,agdi,agdl,agdg,agdh

          if(agda.ne.1.0d01)go to 330
          clm(1)=0.35961113d-07
          clm(2)=0.18246786d-05
          clm(3)=0.22207398d-05
          clm(4)=clm(3)
          clm(5)=0.37098633d-06
          clm(6)=clm(5)
          clm(7)=0.22580118d-06
          clm(8)=clm(7)

          slm(1)=0.24286442d01
          slm(2)=0.57568525d01
          slm(3)=0.12107857d00
          slm(4)=slm(3)
          slm(5)=0.96916794d0
          slm(6)=slm(5)
          slm(7)=0.11169656d01
          slm(8)=slm(7)

          te(7,1)=0.0d0
          te(7,2)=0.0d0
          te(7,3)=1.0d0
          te(7,4)=-1.0d0
          te(7,5)=1.0d0
          te(7,6)=-1.0d0
          te(7,7)=1.0d0
          te(7,8)=-1.0d0
          te(8,1)=1.0d0
          te(8,2)=2.0d0
          te(8,3)=1.0d0
          te(8,4)=1.0d0
          te(8,5)=2.0d0
          te(8,6)=2.0d0
          te(8,7)=3.0d0
          te(8,8)=3.0d0

          feta=1.0d0+eta
          f1=1.0d0+f(8)
          f2=1.0d0-f(8)
          f3=1.0d0+3.0d0*f(8)
          f4=1.0d0-3.0d0*f(8)
          f152=f(15)*f(15)
          r1=f(2)*f(11)
          r2=f(2)*f(12)-c(12)

          tf(1)=-1.5d0*f(8)*f(15)
```

```
        tf(2)=1.5d0*f152
        tf(3)=0.9375d0*f152*f3-0.75d0*f1
        tf(4)=0.9375d0*f152*f4-0.75d0*f2
        tf(5)=1.875d0*f(15)*f1*f4
        tf(6)=-1.875d0*f(15)*f2*f3
        tf(7)=5.625d0*f152*f1
        tf(8)=5.625d0*f152*f2



        tfp(1)=-1.5d0*(f(8)**2-f152)
        tfp(2)=3.0d0*f(8)*f(15)
        tfp(3)=f(15)*(1.875d0*f(8)*f3-2.8125d0*f152+0.75d0)
        tfp(4)=f(15)*(1.875d0*f(8)*f4+2.8125d0*f152-0.75d0)
        tfp(5)=f1*(1.875d0*f(8)*f4+3.75d0*f152)
        tfp(6)=f2*(1.875*f(8)*f3-3.75d0*f152)
        tfp(7)=f(15)*(11.25d0*f(8)*f1-5.625d0*f152)
        tfp(8)=f(15)*(11.25d0*f(8)*f2+5.625d0*f152)



        ta=1.0d0/f(2)/f(13)**5
        flp1=6.0d0
        tg=1.0d0/eta2/eta
        tgp=3.0d0*f(5)*tg/eta2

        do 300 i=1,8
           if(i-3)301,302,301
  302      flp1=8.0d0
           tg=tgp/3.0d0
           tgp=(1.0d0+4.0d0*esq)/(eta2**2*eta2*eta)
           ta=ta/f(13)
  301      tai=ta*clm(i)/(r1*te(7,i)+r2*te(8,i))
           te(2,i)=-tai*tf(i)*tg*eta*te(7,i)/f(5)
           te(3,i)=tai*tf(i)*(te(7,i)*f(8)-te(8,i))/f(15)*tg/eta
           te(4,i)=tai*(flp1*f(5)*tg-eta2*tgp)*tf(i)
           te(5,i)=tai*(tf(i)*f(5)*eta/feta*tgp+flp1*tf(i)*tg
     &     +f(15)/f1*tfp(i)*tg/eta)
  300    te(6,i)=tai*tfp(i)*tg/eta

        go to 340
  330   the=osc(6)
        if(kz.eq.0.0d0)the=the-c(12)*(f(9)+h1-c(1))

        do 341 i=1,8
           sint=osc(5)*te(7,i)+the*te(8,i)-slm(i)
           cost=dcos(sint)
           sint=dsin(sint)
           agde=agde+te(2,i)*cost
           agdi=agdi+te(3,i)*cost
           agdl=agdl+te(4,I)*sint
           agdg=agdg+te(5,i)*sint
  341    agdh=agdh+te(6,i)*sint

  340   return
        end
*************************************************************
        subroutine message(d1,d2,d3,mynod,dest,itype)

*   This subroutine is used to join to disjoint variables
*   into a single array to be sent or received in one message
```

```
          implicit real*8 (a-h,m-z)


          dimension msg2(3)

          integer mynod,dest,pid,itype,msglen

          data pid/0/,msglen/8/


          if(itype.eq.0)then
              msg2(1)=d1
              msg2(2)=d2
              msg2(3)=d3
              call csend(mynod,msg2,msglen*3,dest,pid)
          else
              call crecv(dest,msg2,msglen*3)
              d1=msg2(1)
              d2=msg2(2)
              d3=msg2(3)
          endif


          return
          end
*************************************************************
          subroutine cons1

*    This subroutine is used by NAVSPASUR to set the
*    constants used by the satellite motion model

          implicit real*8 (a-h,m-z)

          common /cons/c(25)

          BETA=398597.62579588d0
          ERKM=6378.135d0
          FLAT=298.26d0

*    K-TERMS

          C20=-0.4841605d-03
          C30=0.95958d-06
          C40=0.55199d-06
          C50=0.65875d-07
          c(3)=-0.5d0*C20*dsqrt(5.0d0)
          c(4)=C30*dsqrt(7.0d0)
          c(5)=0.375d0*C40*dsqrt(9.0d0)
          c(6)=C50*dsqrt(11.0d0)

*    TWO PI

          c(7)=6.283185307179586d0
          c(16)=1.0d0/c(7)

*    HERGS/DAY , SECS/HERG, MINS/HERG

          c(9)=ERKM*dsqrt(ERKM/BETA)
          c(8)=86400.0d0/c(9)
          c(17)=1440.0d0/c(8)
```

```
*    WE(RAD/DAY), WE- 2 PI, WE(RAD/HERG)

          c(11)=6.3003880987d0
          c(10)=c(11)-c(7)
          c(12)=c(11)/c(8)

*    EARTH FLATTENING

          c(13)=(2.0d0-1.0d0/FLAT)/FLAT

*    SM/ER, KM/ER, NM/ER

          c(20)=ERKM/1.609344d0
          c(21)=ERKM
          c(22)=ERKM/1.852d0

*    DEG/RAD
          c(23)=360.0d0/c(7)

*    RANGE RATE/ER/HERG TO CYCLES/SECOND - CONVERSION
          c(24)=c(21)*216.980d+06/(c(9)*2.997925d+05)

*    FENCE PLANE DISPLAC FROM EARTH CENTER
          c(25)=0.31000d-02


          return
          end
*************************************************************
```

APPENDIX D

P³T SOURCE CODE LISTING

This appendix contains the listing of host and node programs comprising **P³T** program set. The host program loads the node program and clears the nodes once the process is complete. The node program contains the instructions for the nodes to complete their respective portions of the **P³T** parallel algorithm. The node program assigns Node 0 to be the distributing node, the highest numbered node to be the collecting node, and the remaining nodes to be the working nodes. The working nodes execute the original NAVSPASUR subroutine **PPT2** with only minor modifications.

Host Program:

```
      program P3th

*     This host program loads the node program P3tn on the
*     nodes of the attached hypercube.  Upon completion of
*     the catalog of satellite data, program clears nodes
*     for another process.

*     Set host specific parameters

      data pid/0/

*     Set process id

      call setpid(pid)

*     Load program P3tn on the nodes

      print*,'loading nodes'
      call load('p3tn',-1,pid)

*     Receive message that nodes are complete

      call crecv(99,istop,4)
      print*,'nodes complete'

*     Kill process on nodes
```

```
call killcube(-1,-1)

stop
end
```

Node Program:

```
        program P3Tn
*   This program propagates n-2 satellites concurrently,
*   where n is the number of nodes belonging to the
*   attached cube.  Node 0 is the distributing node and
*   Node n-1 is the collecting node.  The remaining nodes
*   are the working nodes that propagate the satellites
*   using NAVSPASUR subroutine PPT2.  For simplicity,
*   the tasks for all nodes are combined on this one node
*   program.  Tasks are partitioned by logical if statements.

      implicit real*8 (a-h,o-z)
      real*8 kf(10)

      integer mynod,numnode,hostid,pid
      integer mynode,numnodes,myhost,mclock
      integer eof

      dimension ar(6,8),br(3,6),cr(3,6),dv(3)

      common/cons/a(64)

     common/ppt/f(25),osc(10),kf(10),cf(10),bs(3,4),u(3),v(3),w(3),r,
     &     vel(3),dind,tm,dkz,dident
      common/dcsub/pe(6,8),e(8,8),ep(8,8),g(8),gp(8),ifti(8),ifto(8),
     &     iteri,itero,jof,jol,stat(20),tol(6),iw,of(11),ow(8,8)
      common/foreo/rho(3),ros,hdr,hdv,rdv,del,iter
      common/bloc/sat(84,4800)

      data istop/1/,pid/0/,msglen/672/
      data isat/1/,n/1/

      mynod=mynode()
      numnode=numnodes()
      hostid=myhost()

*0000000000000000000000000000000000000000000000000000000000
*   Node 0 reads and distributes data among the working
*   nodes

        if(mynod.eq.0)then

*   Read complete catalog of satellite data

           open(unit=10,file='/usr/phipps/in',form='unformatted')
           read(unit=10,iostat=eof)(sat(j,1),j=1,84)

 1200      if(eof.ge.0)then
             isat=isat+1
             read(unit=10,iostat=eof)(sat(j,isat),j=1,84)
           go to 1200
           endif
           close(unit=10)

*    Send number of data sets to all nodes

        t0=mclock()
        isat=isat-1
        call csend(mynod,isat,4,-1,pid)
```

```
*    Distribute satellite data sets to working nodes

          iter=isat/(numnode-2)
          do 1201 j=1,iter
          do 1201 i=1,numnode-2
             call csend(mynod,sat(1,n),msglen,i,pid)
 1201     n=n+1

          iter=mod(isat,numnode-2)
          n=n-1
          do 1202 j=1,iter
 1202     call csend(mynod,sat(1,n+j),msglen,j,pid)

*    End Node 0

*0000000000000000000000000000000000000000000000000000000000
*WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW

*    Begin working nodes

          else
          if(mynod.lt.numnode-1)then

*    Use subroutine cons1 to set constants

            call cons1

*    Receive number of data sets from Node 0 and compute total number
*    of satellites to propagate


            call crecv(0,isat,4)
            t0=mclock()
            if(mynod.le.mod(isat,numnode-2))then
              iter=isat/(numnode-2)+1
            else
              iter=isat/(numnode-2)
            endif

*    Receive satellite data, execute PPT2, and send results to
*    collecting node

             do 1220 i=1,iter
                call crecv(0,f,msglen)

*    Set parameters for subroutine ppt2

                ind=1
                kz=idint(dkz)

*    Compute secular recovery

                call ppt2(ind,kz)

*    Compute subsequent task, ie. predict position, update elements ...

                ind=idint(dind)
                call ppt2(ind,kz)

 1220     call csend(mynod,f,msglen,numnode-1,0)
```

```
*    End Working Nodes

*WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
*    Begin Collecting Nodes

        else

*    Receive total number of satellite sets

         call crecv(0,isat,4)

*    Collect results from Working Nodes

         do 1230 i=1,isat

 1230          call crecv(-1,sat(1,i),msglen)

*    Write results to external file

         open(unit=11,file='/usr/phipps/out',form='unformatted')
         do 1231 i=1,isat
 1231    write(unit=11,*)(sat(j,i),j=1,84)
         close(unit=11)

*    Send message to Host that process is complete

        call csend(99,istop,4,hostid,pid)

*    End Collecting Node

*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

        endif
        endif

        stop
        end
```

LIST OF REFERENCES

Amdahl, G., "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities", *AFIPS Conference Proceedings*, v. 30, pp. 483 - 485, April 1967.

Arshi, S., and others, "Application Performance Improvement on the iPSC/2 Computer", *A Technical Summary of the iPSC/2 Concurrent Supercomputer*, INTEL Corporation, pp. 17 - 22, 1988.

Bate, R., Mueller, D., and White, J., *Fundamentals of Astrodynamics*, Dover Publications, Inc., 1971.

Brouwer, Dirk,"Solution of the Problem of Artificial Satellite Theory without Drag", *Astronomical Journal*, 64, pp. 378 - 397, 1959.

Danby, J. M. A., *Fundamentals of Celestial Mechanics*, William Bell, Inc., 1989.

Flynn, M. J., "Very High-speed Computing Systems", *Proceedings of the IEEE*, v. 54, pp. 1901 - 1909, December 1966.

Hwang, Kai and Briggs, Faye A., *Computer Architecture and Parallel Processing*, McGraw-Hill, Inc., 1984.

iPSC/2 Programmer's Reference Manual, INTEL Corporation, Order Number 311708-003, June 1990.

iPSC/2 User's Guide, INTEL Corporation, Order Number 311532-006, June 1990.

iPSC/2 VAST User's Guide, INTEL Corporation, Order Number 311571-002, February 17, 1989.

Knowles, Stephen H.,"Accuracy Factors in Satellite Position Determination", *Proceedings of the 1992 Space Surveillance Workshop*, Massachusetts Institute of Technology, Lexington, MA, pp. 218 - 227, April 7 - 9, 1992.

Lyddane, R. H.,"Small Eccentricities or Inclinations in the Brouwer Theory of the Artificial Satellite", *Astronomical Journal*, v. 68, pp. 555 - 558, 1963.

Quinn, Michael J., *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, Inc., 1987.

Solomon, Daniel, *THE NAVSPASUR Satellite Motion Model*, Naval Research Laboratory, Contract N00014-87-C-2547, August 8, 1991.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Techical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Chairman, Department of Mathematics Naval Postgraduate School Monterey, CA 93943	1
4. Commander U. S. Naval Space Command Dahlgren, VA 22448-5170	1
5. Professor B. Neta, Code MA/Nd Department of Mathematics Naval Postgraduate School Monterey, CA 93943	2
6. Professor D. A. Danielson, Code MA/Dd Department of Mathematics Naval Postgraduate School Monterey, CA 93943	1
7. Dr. Stephen H. Knowles U. S. Naval Space Surveillance Center Analysis and Software Department Special Projects Division, Code 837 Dahlgren, VA 22448	1
8. Professor R. Panholzer, Code EC/Pz Chairman, Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943	1
9. Professor B. Russak, Code MA/Ru Department of Mathematics Naval Postgraduate School Monterey, CA 93943	1

10. Dr. Shannon Coffey 1
Mathematics and Computer Technologies Section
Naval Research Laboratory
Naval Center for Space Technology, Code 8242
Washington, DC 20375-5000

11. CPT Warren E. Phipps, Jr. 2
Department of Mathematical Sciences
United States Military Academy
West Point, NY 10996